PROGRAMMING IN THE PROTECTED MODE

July/August 1990 Volume 8.4

Programmer's Journal™

$4.95 Canada $5.95

Running Unprotected
Virtual-8086 Mode
page 34

Converting to a 386 DOS Extender
page 42

OS/2 Memory Models for the 386
page 49

Deliberately Breaking the Rules
page 20

Implementing C++ Features in C
page 60

MAR/APR 91
318542
RON K UNZ
WALL STREET ANALYTICS INC
33-41 85TH ST
JACKSON HEIGHTS NY 11372

REPRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

VARs & Developers
Bring X-Windows Power to DOS
Sign up now for our DESQview API Conference
August 13-16

...w you can bring the power of DESQview to Clipper,™ dBASE™ and Basic.

DESQ view Clipper Library
DESQ view dBASE Library
DESQ view Basic Library
DESQ view Pascal Library
DESQ view API C Library
DESQ view API Debugger
DESQ view API Panel Designer
DESQ view API Reference

Quarterdeck can provide everything you need to put DESQview capabilities into your applications.

There's a growing realization among programmers that for many users, the ideal PC of the future doesn't look all that different from the PC of today. Experts agree that extended DOS will be the dominant operating system for the foreseeable future.

And DESQview is the best way to create powerful, multitasking solutions for the 30 million DOS PCs in use today.

The API Advantage

Programmers who take advantage of DESQview's API (Application Program Interface) get easy access to the powerful capabilities built into DESQview—multitasking, windowing, intertask communications, mailboxes, shared programs, memory management, data transfer, mousing, menu-building and context-sensitive help. In short, you get all the important benefits promised by OS/2. Without the overhead burden and performance trade-offs.

Interactive tools like this Panel Designer make implementing the DESQview API a snap.

API capabilities are now available to Clipper, Basic and dBASE users as well as Pascal and C programmers.

Bells and Whistles

A program taking advantage of the DESQview API can spawn subtasks for performing background operations or new processes for loading and running other programs concurrently. It can schedule processing after an interval or at a certain time. It can use DESQview's intertask communications to rapidly exchange data between programs, share common code and data; or interrupt at critical events. It can use DESQview's menuing and mousing capabilities to create menus. And there's lots more it can do.

80386 Power

Programmers can take advantage of the 80386's protected mode for large programs, yet run on DOS and multitask in DESQview—side by side with other 80386 and DOS programs. The breakthroughs that make it possible: DOS Extenders from PharLap, Rational Systems, Eclipse and Oracle and DESQview support of these extenders.

A Complete Interactive Toolkit —Including a Debugger

API Reference Manual The key to the power of the DESQview API, this manual contains all you need to know to write Assembly Language programs that take full advantage of DESQview's capabilities. And there's an 'include' file with symbols and macros to aid you in development.

API Clipper Library Add new power to client-server database applications. Define tasks that monitor or control any other application; run reports, sorts or numerical analyses in the foreground or background; create custom interfaces with control over keyboard, mouse and window functions. And it's all easy to accomplish because you can treat the entire set of over 300 object-oriented API functions as prewritten UDFs.

"With the DESQview API, you can create programs with abilities far beyond those of Clipper alone... obviously, the uses are limited by your imagination"
—Data Based Advisor, December 1989

You get the API Reference Manual and source code for the library with the Clipper Library package.

API dBASE Library All API functions are available within dBASE III and IV. We include source code for the library, the API Reference Manual and example programs.

API Basic Library The Basic library provides interfaces for the entire set of API functions. Included are the API Reference Manual, source code for the library and example programs.

API Pascal Library The Pascal library provides interfaces for all API functions. It supports Borland Turbo Pascal 4.0 and 5.0 compilers and Microsoft Quick Pascal. Included are the API Reference Manual, source code for the library and example programs.

API C Library You get C language interfaces for the entire set of API functions. It supports the Lattice™ C, Metaware™ C, Watcom C, Zortech C, C++, Microsoft® C, and Turbo C compilers for all memory models. Included with the C Library package is the API Reference Manual and source code for the library.

API Debugger The DESQview API Debugger is an interactive tool enabling the API programmer to trace and single step through API calls from several concurrently running DESQview-specific programs. Trace information is reported symbolically along with the program counter,registers, and stack at the time of the call. Trace conditions can be set to report only calls of interest

API Panel Designer This interactive tool helps you design windows, menus, help screens, error messages, and forms. It includes an editor that lets you build an image of your panel using simple commands to enter, edit, copy, and move text, as well as draw lines and boxes. You can then define the characteristics of the window that will contain the panel: its position, size, and title. Finally, you can specify the locations and types of fields in the panel.

The Panel Designer automatically generates all the DESQview API data streams necessary to display and take input from your panel. These data streams may be grouped into panel libraries and stored on disk or as part of your program.

Some of the applications under development right now using DESQview API Tools: CAD, medical systems, insurance, 3270 network management, mainframe communications, real estate management, typesetting, point of sale, education, stock and commodity trading and online voting.

More Tools are Coming If you'd like to know all the things we're up to in the API Toolkit, plan to attend our API Conference this Summer. Call or write for details.

But for now, we can tell you that we are committed to adding tools as needed by our users. In the works, we have many new, exciting tools that will keep DESQview API users at the forefront of interface development.

Quarterdeck

Quarterdeck Office Systems, 150 Pico Blvd., Santa Monica, CA 90405 (213) 392-9851 Fax: (213) 399-3802

Trademarks: Clipper, Nantucket; dBASE, Ashton-Tate; OS/2, IBM; PharLap, Rational Systems, Eclipse, Eclipse; Oracle, Lattice, Metaware, Watcom, Zortech, Microsoft, Turbo C
©1990 Quarterdeck Office Systems

Circle **147** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Force Fits!

C Programmers: Finally, A Structured Database Language Designed for the Way You Program.

You program in C because it's a powerful, non-interpretive, structured environment. Perfect for the complex, advanced applications you create.

That's why we designed Force with the same powerful language elements that make C so effective.

Force is a structured, non-interpretive dBASE language compiler. Force lets you manipulate your database engine *directly* through a language designed around the engine itself.

You don't have to waste time managing your database engine through a library or a "canned" C solution. Force handles the entire design and management process.

Force produces individual, executable files that can be distributed royalty-free, without a runtime interpreter. And Force eliminates the memory fragmentation and management problems that are common in interpretive database systems like Clipper.

In short, Force is the first database programming environment designed to perform in the world you work in every day.

Increase Your Options.

The great thing about C programming is the multiple high-level languages available to you for solving various kinds of problems.

The weak link has always been database programming. The unstructured, interpretive environment of dBASE and Clipper is inadequate. C database libraries take too much time and effort to manage.

Force is different. It's a database tool that can handle the most complex C applications. And since you control your engine directly, Force speeds the entire development process.

Like a C compiler, the Force compiler does a lot of the work for you. It validates your syntax. It generates native code. It makes linking easy. It lets you use your favorite C and assembler routines *without* an extend system (just link it). It supports conditional compilation, includes files, and defines.

Force frees you to create database programs just the way you create other kinds of solutions.

Complete Network Support.

Force offers complete network support, as well as an advanced method of record-locking.

For maximum productivity, Force can create a distributed processing environment where portions of large tasks can be assigned to several network nodes, then collated back at the file server.

More Than a Function Library.

Force is a development environment with features and enhancements never before available in a database programming tool:

- C-style Pointers, Structures and Variables
- Compatible with Hundreds of C Programming Libraries
- Instant TSRs: Convert dBASE programs into TSR utilities
- No Runtime Interpreter: Smaller EXEs
- Full Syntax Checking at Compile Time
- Complete Menuing System
- Multiple Data Types

Force is Faster.

Force has the power to handle programming tasks much faster than using C itself:

- A MAKE utility that can handle an unlimited number of dependencies
- Any Directory Manager
- An Accounting Package
- Any Problem Needing List Management
- A Sales Tracking System, Report, or Label System
- The List Goes On and On!

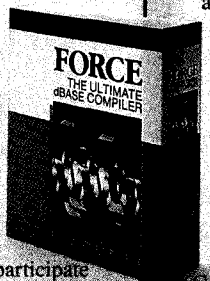
A Fitting Guarantee.

Force is unconditionally guaranteed to be the best database programming tool you've ever used. If you're not completely satisfied, return it within 90 days and we'll gladly refund the $695 purchase price.

Order your copy of Force *now*. Call (800) 922-3001, or return the coupon.

Trademark/Owner: Force/Sophco; dBASE/Ashton-Tate; Clipper/Nantucket Corp.

FREE SuperMake!



Order Force now and get a FREE copy of SuperMake—the first distributed make utility that lets every node on your network participate in completing big jobs.

Written in Force and designed exclusively for micros, SuperMake turns monumental builds into manageable tasks. And you can keep SuperMake, even if you return Force!

A $199 Value—Absolutely Free!

Force Fits My Needs! Please send me _____ copy/copies of Force 2.1 at only $695. If I'm not completely satisfied, I know I can return Force within 90 days for a full refund!

Name _____

Company _____

Address _____

City _____ St. ___ Zip _____

Payment: ☐ Visa ☐ MC ☐ Amex ☐ P.O.

Card/P.O. Number _____

Exp. Date _____

Auth. Signature _____

Sophco

4730 Walnut, Building B, Boulder, CO 80301

(800) 922-3001

(303) 444-1542

Germany: KRS (02174) 679-0
United Kingdom: CTL 0372 727269
Australia: TALSOFT (02) 317-4404

PROVIDED BY DBASEWIZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Programmer's Journal™

July/August 1990 Volume 8.4

FEATURES

ON GRAPHICS **FASTER ELLIPSES AND MORE** *By Michael Abrash* 24
Installment 19 in which Michael works his VGA magic on faster ellipse-drawing along with covering EGA color mapping and a new book on the 8514/A

PROTECTED MODE **RUNNING UNPROTECTED** *By Dean C. Wills* 34
Exploring 80386 system programming by setting up a spiffy virtual-8086 monitor

PROTECTED MODE **CONVERTING TO 386 PROTECTED MODE** *By Ronald Berry* 42
Specific guidelines for porting an application to run under a 386 DOS extender

PROTECTED MODE **THE WORLD IS FLAT** *By Brett Glass* 49
A look at the OS/2 2.0 memory model and other new features in the 32-bit version

PROTECTED MODE **FORTRAN ON THE 386** *By M. Steven Baker* 53
386 DOS extender development tools, focusing on FORTRAN

C **USING C++ FEATURES IN C** *By Allen Holub* 60
Allen has fun with the C preprocessor implementing a "stack" abstract data type

C **DIFFERENT TIMES** *By Murray Lesser* 67
Some functions and a system utility for correcting drift in the real-time clock

ASM **ERRORS AND STREAM I/O** *By John Otken* 74
John develops some useful error handling modules along with a fast stream I/O library for assembly language programmers

COBOL **COBOL-85 NESTED** *By Richard D. Riehle* 83
Using features added in the COBOL-85 standard to create simpler programs

COLUMNS

REDIRECTIONS **PINK NOISE** *By Frank D. Greco* 11

CONSULTING **DELIBERATELY BREAKING THE RULES** *By Paul Barkley* 20

IMPRESSIONS **386 DOS EXTENDERS** *By M. Steven Baker* 91

OFFRAMP **JAPANESE VCRS AND DRAMS** *By Hal W. Hardenbergh* 98

DEPARTMENTS

EDITOR'S NOTE 4

LETTERS 6

PJ PRESSROOM 88

PJ BILLBOARD 93

ADVERTISER INDEX 95

COVER

The 386 was designed "to protect and serve" programming needs. So whether you're breaking the rules or religiously converting to new ones, this issue will help you adhere to the laws of protected mode programming.

Artist: Rob Schuster

Articles and letters in *Programmer's Journal* represent the views of the authors and not necessarily those of the publisher. Letters to the editor become the property of *Programmer's Journal* and may be edited for publication.

Note that all code and listings in *Programmer's Journal* appear "as is," and *Programmer's Journal* makes no guarantee or warranty of their quality, performance, accuracy, or suitability for any particular purpose. *Programmer's Journal* assumes no responsibility or liability to any person or entity with respect to any damage caused or alleged to be caused by their use.

Advertising: For ad rates, deadlines, and placement, call Robert Tappan or Jay Moore at 800-234-0386, or write to *Programmer's Journal*, P.O. Box 70167, Eugene, OR 97401-0110. *Programmer's Journal* is produced using the following software products: Ventura Publisher,™ Aldus PC PageMaker,® and Corel Draw. The PostScript™ files are output on a Linotronic™ 200.

Programmer's Journal (ISSN 0747-5861) is published bimonthly by Oakley Publishing Company, 150 N. 4th Street, Springfield, OR 97477-5454. Second-class postage paid at Springfield, OR, and additional mailing offices. USPS # 000293. **POSTMASTER:** Send address changes to *Programmer's Journal*, PO Box 70387, Eugene, OR 97401-0121.

© 1990 OAKLEY PUBLISHING COMPANY

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Another New Product from the makers of Soft-ICE & MagicCV

ANNOUNCING BOUNDS-CHECKER

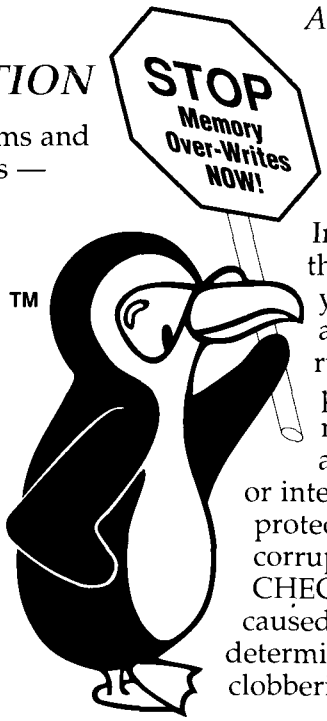
Gives you the protection of a protected operating system under MS-DOS.

Finds out-of-bounds memory accesses — AUTOMATICALLY.

AUTOMATIC PROTECTION

Flush out those Nasty pointer problems and other out-of-bounds memory accesses — AUTOMATICALLY.

Each time you make a change to a program, run BOUNDS-CHECKER while testing the new code. If you accidentally access out-of-bounds memory, BOUNDS- CHECKER will pop up displaying the offending SOURCE LINE. And your program runs at full speed.



Increase Productivity During Development, Improve Reliability During Q/A

You can run BOUNDS-CHECKER while testing your program. There are no additional steps to your testing cycle, but you can feel secure when the program has passed through BOUNDS-CHECKER with no reported problems.

Many over-write problems and other out-of-bounds memory accesses do NOT show up during normal testing. An out-of-bounds memory location may be modified, but that particular location doesn't happen to be important at the time. Once the program is in the field and a certain network is loaded or a certain T&SR or device driver is loaded, that memory location suddenly becomes very important... AND THE SYSTEM CRASHES.

You can prevent these problems by making BOUNDS-CHECKER a standard part of your testing procedure.

HOW IT WORKS

BOUNDS-CHECKER uses the 80386 virtual machine technology to provide real-time memory protection. In addition BOUNDS-CHECKER uses the symbolic information output by your compiler to differentiate CODE and DATA. When your program is running, BOUNDS-CHECKER protects the program's CODE and all memory outside your program. When an MS-DOS system call, BIOS call or interrupt occurs, BOUNDS-CHECKER protects the system software from corrupting your code. So, BOUNDS-CHECKER will not only detect problems caused by your program, it will also determine if a T&SR or other program is clobbering you.

Enormous Productivity Gain

Don't spend days or even weeks chasing down one of those subtle memory over-write problems. Use BOUNDS-CHECKER to pinpoint the problem in *seconds*.

Don't RISK a Recall

Don't take the unnecessary RISK that your program is unknowingly clobbering an out-of-bounds memory location. BOUNDS-CHECKER was developed to find these potential time bombs quickly, painlessly and AUTOMATICALLY!

$249.00

Requires 80386 PC.
MS-DOS is a trademark of Microsoft Corporation.

Nu-Mega TECHNOLOGIES

CALL TODAY (603) 888-2386 or FAX (603) 888-2465

P.O. BOX 7607 ■ NASHUA, NH ■ 03060-7607 ■ U.S.A.

Circle 160 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EDITOR'S NOTE

In this issue of *PJ* we focus on programming in the protected mode of the Intel 80x86 CPUs. Over the years, DOS operating system software has lagged significantly behind hardware innovations. EMS (Enhanced Memory Specification) let us finally access memory above DOS's 640K limit, followed by XMS (eXtended Memory). IBM and Microsoft promised a protected mode version of DOS (ADOS, DOS 5.0) years ago. The new DOS turned out to be OS/2, and versions of 16-bit OS/2 are still stabilizing. The original target for OS/2 was the 80286, but since then Intel has developed and produced first the 80386 and now the 80486. The 80586 won't be far behind.

OS/2 version 2.0 for the 80386 is "under development," meaning that Microsoft will sell developers a Software Development Kit, but it's not likely to become a stable product until 1991. Right now, IBM and Microsoft seem to have differences of opinion on OS/2 for the 286. IBM is still promising large corporations and end-users a DOS for all those IBM PC/AT and PS/2 machines with 80286 chips. Microsoft, always more technology-driven than IBM, in some ways would rather forget about 16-bit OS/2 and move on to the true 32-bit version.

IT'S A WINDOWS WORLD

The hot news is the long-awaited arrival of Microsoft Windows 3.0 which contains its own protected mode 386 DOS extender. Windows 3.0 is a complex product with several different modes that are sensed at runtime. Of interest to programmers is the *386 enhanced* mode that uses Microsoft's DOS extender.

Another significant feature of Windows 3.0 is its support for DOS Protected Mode Interface (DPMI). DPMI is a software specification designed to allow 286 and 386 DOS extender programs to run under other protected-mode multitasking programs such as Windows/386, DESQview, and VM/386. In a way, DPMI can be considered a parallel extension to VCPI (Virtual Control Program Interface) which allows DOS extender programs to multitask and co-exist with EMS software. Surprisingly, Intel has been the main proponent of DPMI, convincing Microsoft, Quarterdeck, Phar Lap, Rational, IGC, Ergo and others to play ball.

Not to be outdone, Quarterdeck has announced that DESQview will incorporate X Window, supporting both OSF's Motif and UI's Open Look user interfaces. This may provide a powerful alternative to Windows and Presentation Manager as a graphical, multitasking user interface under DOS.

THE HERE AND NOW—DOS EXTENDERS

While OS/2 was delayed, DOS extenders for both the 80286 and 80386 allowed developers to write code that used features of these new CPUs. DOS16/M (Rational Systems) and OS/286 (now Ergo Computing, formerly A.I.Architects and Eclipse) provided a solution for the 286; Phar Lap, Ergo, and IGC provided 32-bit DOS extenders for the 386.

Many of the newly-arriving development tools incorporate DOS extenders in enhanced versions of existing products. Examples include Borland's new C++ environment, Zortech's C++ v2.1, Lahey's F77/L32 v3.0, MicroWay's and Watcom's C/386, and FORTRAN/386 compilers. In addition, libraries are becoming available for use with DOS extenders.

The high-end CAD market has provided the protected mode its greatest success. 386 DOS extenders have virtually swept the market for CAD. The program development tool market may well be next.

Steven Baker

Programmer's Journal™

Editor
M. Steven Baker

Technical Editor
Colin Elligsen

Associate Editors
Cecelia Hagen
Deborah Healey

Contributing Editors
Michael Abrash
Frank D. Greco
Tom Swan

Publisher
Elizabeth Oakley

Associate Publisher
Bobbi Sinyard

Production
Wanda Walker

Illustration
Jim Carpenter

Advertising
Robert Tappan
Jay Moore

Circulation
Eric Malin

Subscriptions
Pamela Coleman

Accounting
Jill Edwards

Office Manager
Sylvia Long

Business and Editorial Offices
Programmer's Journal
150 North 4th Street
Springfield, OR 97477-5454
(503) 747-0800
(800) 234-0386
Fax: (503) 746-0071

Subscriptions
1 year (6 issues) $24.00

Change of Address
Programmer's Journal
PO Box 70387
Eugene, OR 97401-0121

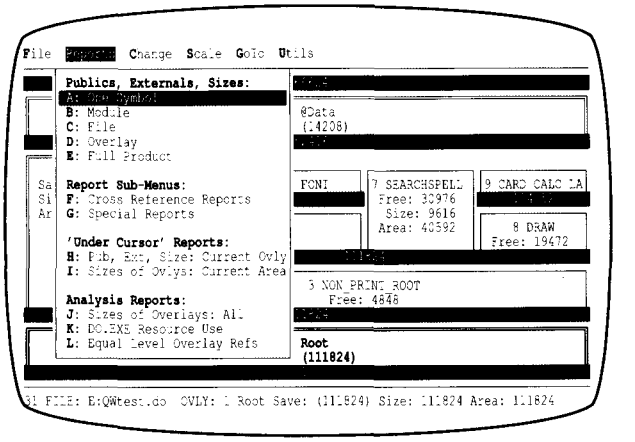
© 1990 OAKLEY PUBLISHING COMPANY

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

OVERLAY DESIGNER™

for Plink86

and RTLink



W Y S I W Y L

What You See Is What You Linked

Save Programmers' Time

You can expect to save up to 75% of the time you now spend analyzing and testing overlay structures and still do a better job. Overlay Designer is a state-of-the art modeling program to help programmers design and analyze the overlay structures of large DOS application programs. Overlay Designer automatically reads the .MAP file, .OBJ files, .LIB files, and (for RTLink only) the monitor output captured to file during the link, and builds a database describing the link. The programmer then uses Overlay Designer's graphic, algorithmic, and tabulation tools to design a more compact, fast, and reliable overlay structure. Overlay Designer is ***Computer Aided Design*** for DOS overlays. Overlay Designer has been beta tested for 9 months on products ranging in sales from one million dollars per year to one million dollars per month.

Fast Results - Almost No Setup

Overlay Designer automatically reads its data from .MAP, .OBJ, and .LIB files and (RTLink only) the monitor output capture file. You can select 3 types of information to import into the Overlay Designer file: 2-dimension overlay diagram, detailed memory sizes, and symbol references. The diagram import takes only a few seconds. One beta test programming team posted a 3 ft x 4 ft overlay diagram on the wall each Friday. The full import takes less time than the link.

A Fully Professional Software Package

Overlay Designer consists of over 120,000 lines of source code, developed and supported by the author of Spellguard (® Computer Associates, Infoworld Product of the Year 1981), Display Manager (® Digital Research), and primary author of Q&A (® Symantec Corp., InfoWorld Product of the Year 1987, 1988, PC Magazine Technical Excellence Award 1988). It is fully documented with an indexed manual and has context sensitive help at every input screen.

Benefits

- Time saved by programmers gives quick payback of price
- Diagrams expedite communications in multi-person projects
- Improve morale by automating tedious detail work
- Automatic technical documention of overlays
- Reduce QC testing; many overlay bugs found automatically
- Meet project deadlines by reducing overlay design time
- Improve your product with more compact, reliable overlays

Features

- Detection of direct "equal level" overlay calls
- 35 dynamic tables to cross reference symbols & their use
- Separation of references into the categories *code* and *data*
- Scalable screen and hard copy diagram of overlay structure
- Free area sizes and potential memory reductions on diagram
- "What if " testing of moving memory between overlays
- Undo of tests of moving memory between overlays
- Sort operations on tables to identify important relationships
- Filter operations on tables to eliminate extraneous data
- Memory size reports by segdef, module, file, & overlay
- Zoom on table cells to access reports on the contents of cell
- Memory sizes reported in hexidecimal or in decimal
- Run your editor program without leaving Overlay Designer
- Make Overlay Designer Database in "build" .BAT file
- Stack report tables to a level of 12 deep
- Immediate hard copy of all or part of any table
- Cross reference publics & externals by file, module, overlay

"After my first day with Overlay Designer, I had done work that would have otherewise taken me 2 weeks."
- Mike Milsner, Senior Programmer, Symantec Corp.

COLEMAN SOFTLABS INC

Requires 512K MSDOS, Hard Disk,
and Plink86 or RTLink.
Price 1,200.

Plink86 is a trademark of Polytron/Sage Software.
RTLink is a trademark of Pocksoft, Inc.

Coleman Softlabs, Inc.
296 Bay Road, Dept A,
Atherton CA 94027
Phone (415) 322-9006
Fax (415) 322-1622

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS

EMPHASIS ON ASSEMBLY

I recently renewed my subscription to *PJ* after it had lapsed for a few months. At the same time I bought the one back issue I hadn't bought at the store during the time I wasn't subscribing. I wanted you to know that your recently increased emphasis on assembly language is the main reason I re-subscribed.

I gather from the Letters of the back issue that arrived today that there has been some discussion regarding whether assembly language (*pace*, John Otken!) or high-level languages are adequately covered elsewhere: Otken's and Abrash's treatment of assembly language has no parallel. *Dr. Dobb's* and *Computer Language* are fine, but the world doesn't need another one! Keep up the good work!

H. Sanner
Bladensburg, Maryland

APPLES AND ORANGES?

In the RISC-vs.-CISC debate between Andrew Allison and Hal Hardenbergh (Letters, March/April 1990), Hal makes a common mistake made by CISC fans in comparing the 80486 with current RISC implementations. The 80486 came out in late 1989. The 88000, MIPS, SPARC, and other machines are earlier implementations, and newer versions of them will become available in 1990. It isn't fair to compare the sheer performance of the 80486 with the older RISC technology. Instead, let's compare the 80486 with the 80860, a RISC chip which Intel brought out at essentially the same time (actually, somewhat earlier). It is fair to assume that Intel has applied what they have learned about making fast chips when designing the new architecture of the 80860 (as opposed to the 80486, whose primary design concern was compatibility with earlier chips).

Here is a brief comparison in chart form:

	80860	80486*
Components on chip	1.0 million	1.2 million
On-chip cache	4K code/8K data	8K unified
Clock rate	40 MHz	25 MHz
Dhrystones	~85,000	~26,000
Instruction length	4 or 8 bytes	1-10 bytes

Clearly the 80860 does more with less when it comes to raw performance. The gap will only increase. The 80486 and

ILLUSTRATION BY JIM CARPENTER

subsequent 80x86-compatible chips are saddled with a dated instruction set with which they have to be compatible. The 80860 can initiate multiple instructions in different parts of the chip partly because of the design of its instruction set. Although I can imagine an 80x86-compatible chip of the future doing something similar, just think how much silicon would have to be invested. And think about what could have been done with that silicon if it had been spent differently.

The reason RISC has not caught on in PC-land is because PCs tend to run hand-coded, nonportable programs for 80x86s, and RISCs are not compatible with 80x86s. RISC is popular in UNIX-land because UNIX is written in C, so every time you come up with a new RISC architecture you have an operating system and a whole slew of applications ready and waiting as soon as you have a C compiler.

Notably, there is at least one 80486-based PC with a socket for the 80860. Intel offers development environments for the 80860 which run on PCs. This is the optimum tradeoff: Hal gets to run 80x86-specific software on his PC, and performance jocks get a top-of-the-line RISC chip to blow the doors off Hal's 80486.

The price/performance numbers take on a whole new meaning when you add an 80860 to an 80486 box as a co-processor. Such a machine would have an unbelievable price/performance ratio, far higher than that of either of Andrew Allison's UNIX-running 80x86 boxes. The problem faced by this hardware is the same one faced by all RISC boxes: they need software. If that consideration did not exist, neither would the 80486.

Remember when the 8087 first came out? The only people interested in it at first were the CAD/CAM folks. The rest of the world figured they didn't need the power offered by math coprocessors ("My 8088 can do anything I want it to!"). As the coprocessors got more popular, programs began billing themselves as "Able to utilize 8087," and the coprocessors became even more popular. I see no reason for history not to repeat itself—starting, in point of fact, with the CAD/CAM folks, since 80850 has several instructions ready-made for such applications.

* NOTE: Performance characteristics for the 80486 are lifted from Hal's response in this Letters column [issue 8.2 page 16]. Data on the 80486 and 80860 are from Intel literature and "The Intel 80860," Neil Margulis, *BYTE Magazine* Vol. 14, No. 13, December 1989, respectively.

Nicholas Wilt
Durham, North Carolina

Hal Hardenbergh responds: *Me ignore or low-rate the i860? I wrote a 13-page term paper on the i860 for my class in RISC Architecture last year; I've even tried to interest my employer in the use of the i860 as an alternative to the dedicated systolic arrays which are commonly used to perform high-end image processing. Although the i860 is, in my opinion, an exceptionally good numeric and graphics accelerator, it does have its warts. First, existing compiler technology cannot effectively deal with the dual-operand and exposed pipelines of the i860's FP unit. This means one uses compilers and accepts about 10% of the i860's performance capacity, or one hires human beings to write carefully tweaked code. I understand the latter is no longer done.*

Secondly, the integer unit is primarily intended to calculate the address of the next operands to be used by the FP and graphics units. This means the i860 is virtually unusable as the main processor in a computer system (rumor has it that Olivetti is going to give it a try anyhow). More of my thoughts on the i860 appear on pages 130 and 131 of the February '90 Dr. Dobb's, *which were abstracted, with permission, from a private communication.*

486 vs. RISC? Take a look at the May issue of UNIX Review. *Three 486-based machines swamped the 88000-*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

based DG Avion and the MIPS-based DECstation 3100, and flat blew the SPARCStation 1 out of the water, as measured by price/performance (dollars per Khornerstone). Those three RISC machines are the latest low-cost workstations from the heart of RISCland. The latest and greatest RISC machines which Nicholas Wilt alludes to are all $50K-and-up machines and hence will lose any price/performance contest by default.

Oh, yes: a 33MHz 486 does 37k+ Dhrystones using commonly available compiler technology, and Intel's new submicron version of the 486 will do 55k+ Dhrystones at 50Mhz. [See Hal's new "OFFRAMP" column. —Ed.]

HOT GRAPHICS

The March/April issue of *PJ* was great. You're right—graphics is a hot topic across the whole industry. Here are my two cents on the Super VGAs. First, even with the differences, there are many similar Super VGAs, as Chris Howard points out. The VGA vendors shot themselves in the feet when each came out with different variations of 800x600 graphics. Why? Except for Windows, AutoCad, and a few other packages with vendor-supplied drivers, few software houses want to write to each and every spec. Now, after the horse is out of the barn and millions of boards have been shipped, VESA is trying to get the horse back in again.

Tinkering with this stuff on one's own PC or in an environment where one can be sure that the Super VGAs are alike is one thing, but if you want to do a commercial application for Super VGA you have to get one of every card, license an add-on library from Genus or roll your own and qualify each and every video card. VGA was tough enough in the early days, with lots of near BIOS-compatible cards, BIOS-compatible cards, almost register-compatible, etc. However, if I write to the SINGLE VGA specification, I have no real issues with VGA compatibility. I suspect that 800x600 will continue to be somewhat ignored by software developers since there are so many variations of it. VESA 1024x768, TIGA, and 8514/A all stand a chance of emerging as the next popular graphics standard. Possibly they will all survive, simply because the video card manufacturers have had the good sense to

LAN SCHEDULER
(For IBM Token Ring Network)

Unattended LAN based batch processing. Concurrent execution of multiple jobs on designated network workstations. Conditional job scheduling, job priority, schedule window, optional security, sophisticated calendar, no disk trashing during idle time. Various user friendly screens for schedule data maintenance. Log file shows the history of each job execution including captured screens and DOS error messages.

The product can be used on IBM, NOVELL and similar DOS based LANs.
Single workstation license.....$200
Network license.....$750
Add S/H $5 in US, $15 abroad.
MICROWORK
47 W. St. Andrews, Deerfield,IL 60015
(708)-940-8979

Contact this advertiser directly.

Instant-C

INCREMENTAL C COMPILER

"An Utterly Addictive Speed Improvement"

B. Rubin, *The C Gazette*, Autumn 1989

Speed Microsoft™ C development. *Instant*-C is a Microsoft-compatible incremental compiler. It can rebuild 100,000 line programs in seconds. And you can still use Microsoft C's optimization.

Find bugs other compilers miss. *Instant*-C debugs as fast as it compiles. Here are some bugs that (in a Microsoft C program) could hang your computer. *Instant*-C catches them automatically:

- Array reference out of bounds
- Use of uninitialized pointer
- Read/write through null pointer
- Stack overflow
- Clobbering the program or DOS

For those few bugs that still elude detection, *Instant*-C integrates the debugger with the compiler for true source level debugging.

No slowdown on big programs. *Instant*-C uses our DOS/16M technology, so you can work on programs bigger than 640K with no swapping to disk.

Everything you need. *Instant*-C has much more to offer, like dynamic cross referencing and lint-like static checking. Support for third party libraries. Interactive testing of single functions or partial programs. Everything you need to program and debug faster.

Call 508-653-6006 for *free* information.

Rational Systems, Inc. 220 N. Main St., Natick, MA 01760 PJH1

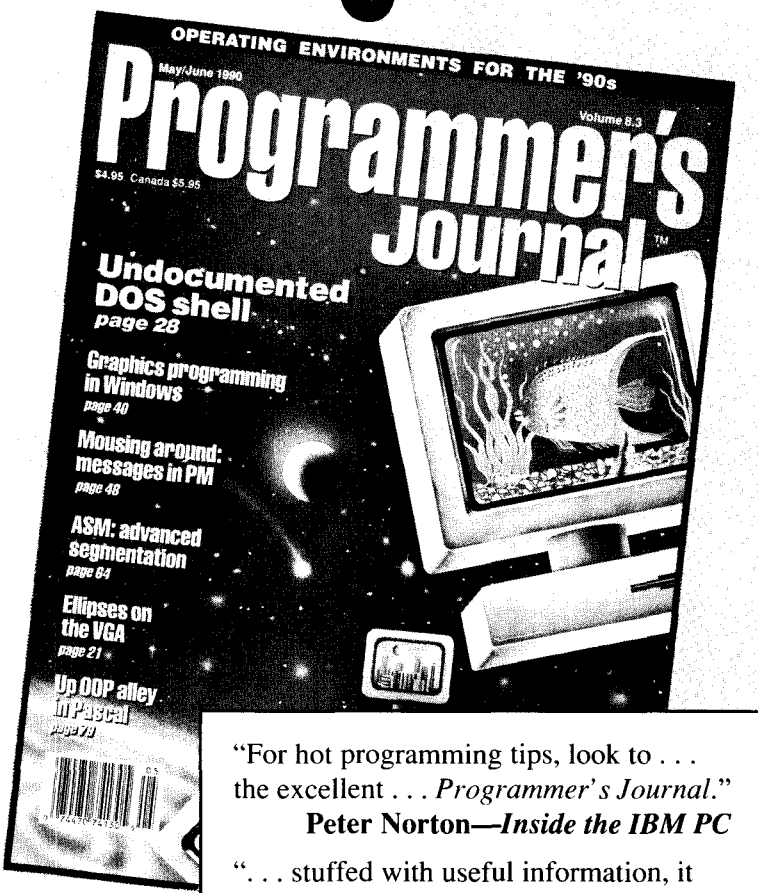
Instant-C requires an 80286/386/486 and at least 1M of memory. Microsoft is a registered trademark of Microsoft Corp.

Circle **20** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

"...love this magazine!" *

Industry experts agree that when it comes to a truly useful programming magazine, ***Programmer's Journal*** just can't be beat. Recognized by the **Computer Press Association** in 1989 for excellence in journalism, ***PJ*** is ". . . full of practical information and guidance presented with superb clarity and perspective." Try ***PJ*** and judge for yourself. It's easy—**just call toll free.** Start receiving 6 issues a year for only $19.95.



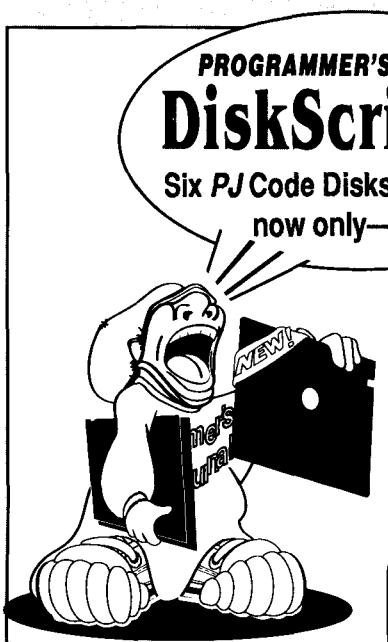
"For hot programming tips, look to . . . the excellent . . . *Programmer's Journal*."
Peter Norton—*Inside the IBM PC*

". . . stuffed with useful information, it definitely deserves a look."
Ray Duncan—*Dr. Dobb's Journal*

* "I really love this magazine!"
Paul Mace—Paul Mace Software

To subscribe call toll free:

800-234-0386



All Our Code—On Disk

Typing in long code listings uses up a lot of valuable time. And calling up bulletin boards is not exactly a free ride. So what does the modern programmer do? Get a ***PJ* Code DiskScription,** of course!

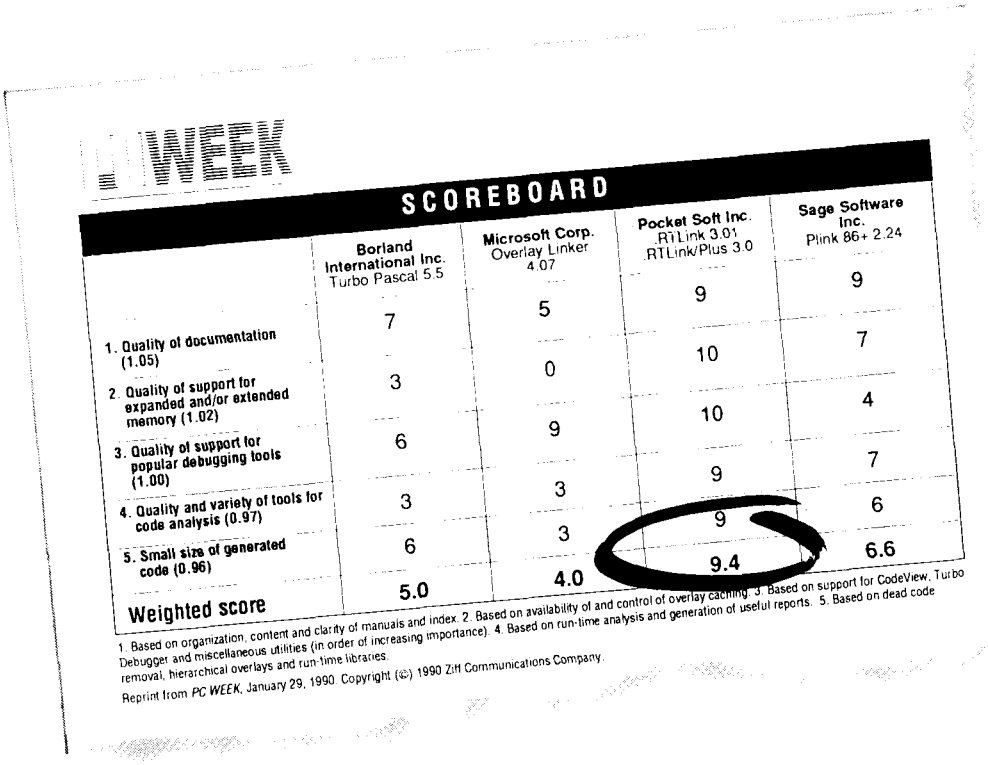
Each disk contains all of the program listings published in ***PJ*** plus .EXE files, .OBJ files, benchmark source files and more! If purchased separately the regular price for a year's worth of ***PJ*** code (6 disks) would be $60.00. **You can save yourself some time, effort and over $20.00 by ordering your *PJ* Code DiskScription today!**

Guarantee: *If you are dissatisfied with your* ***PJ Code DiskScription*** *for any reason, we will gladly send you a full refund on all unmailed diskettes.*

All *PJ* Code DiskScriptions must be prepaid. Return the card on the opposite page with payment. Or, for faster service with your VISA MC call toll-free: 800-234-0386

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

.RTLink/Plus™



PC WEEK

SCOREBOARD

	Borland International Inc. Turbo Pascal 5.5	Microsoft Corp. Overlay Linker 4.07	Pocket Soft Inc. RTLink 3.01 RTLink/Plus 3.0	Sage Software Inc. Plink 86+ 2.24
1. Quality of documentation (1.05)	7	5	9	9
2. Quality of support for expanded and/or extended memory (1.02)	3	0	10	7
3. Quality of support for popular debugging tools (1.00)	6	9	10	4
4. Quality and variety of tools for code analysis (0.97)	3	3	9	7
5. Small size of generated code (0.96)	6	3	9	6
Weighted score	5.0	4.0	9.4	6.6

1. Based on organization, content and clarity of manuals and index. 2. Based on availability of and control of overlay caching. 3. Based on support for CodeView, Turbo Debugger and miscellaneous utilities (in order of increasing importance). 4. Based on run-time analysis and generation of useful reports. 5. Based on dead code removal, hierarchical overlays and run-time libraries.

Reprint from *PC WEEK*, January 29, 1990. Copyright (©) 1990 Ziff Communications Company.

The clear winner against first rate competition.

.RTLink/Plus™ was the clear winner in *PC Week*'s review of overlay linkers. Overlay linkers are used to reduce MS-DOS program memory image size.

PC Week pointed out that .RTLink/Plus added several tricks of its own to expand link time technology to allowing programs to execute in less memory, improve performance, save development time and save substantial amounts of disk space in shipping media and on target machines.

.RTLink/Plus is the only multiple and nested overlay linker which fully supports Microsoft's CodeView© Debugger.

Get Pocket Soft's .RTLink/Plus and get new features which extend any MS-DOS compiler to a new and higher level of productivity.

Send us your name and fax number (or address if you prefer) and we will send you technical specifications on .RTLink/Plus and a reprint of the *PC Week* review.

.RTLink/Plus– Clearly the Best

Pocket Soft, Inc.

Fax **(713) 460-2651**
Phone **(713) 460-5600**

P.O. Box 821049
Houston, TX 77282

Pocket Soft is an authorized licensee of Microsoft®, CodeView® information. "Microsoft" and "CodeView" are registered trademarks of Microsoft Corporation. .RTLINK, .RTLINK/PLUS and Pocket Soft, Inc are trademarks of Pocket Soft, Inc. Turbo Pascal is a registered trademark of Borland International, Inc. Plink86+ is a registered trademark of Phoenix Technologies, LTD.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

STANDARDIZE on the programmatic interfaces. The one that emerges as the winner will be the one providing the best combination of speed and ease of programming for the money.

I enjoyed Mike's VGA article, as always. I have a VGA graphics benchmark suite that tests the speeds of VGA cards doing normal graphics operations, and tests screen fills using write modes 0, 1, and 2. Mike has inspired me to add a write mode 3 test. The hottest VGA cards I've seen are Ahead Systems' VGA Wizard and Cardinal's VGA 600. But, as of this writing, the prize goes to the VGA embedded in the system board on the Compaq SystemPro. It fills the screen at 26.5 million bits per second—a real screamer—six times faster than IBM's VGA on the Model 70.

Mike, what are you doing in Burlington, VT? Working on a project with the Vermont company that produces graphics cards?

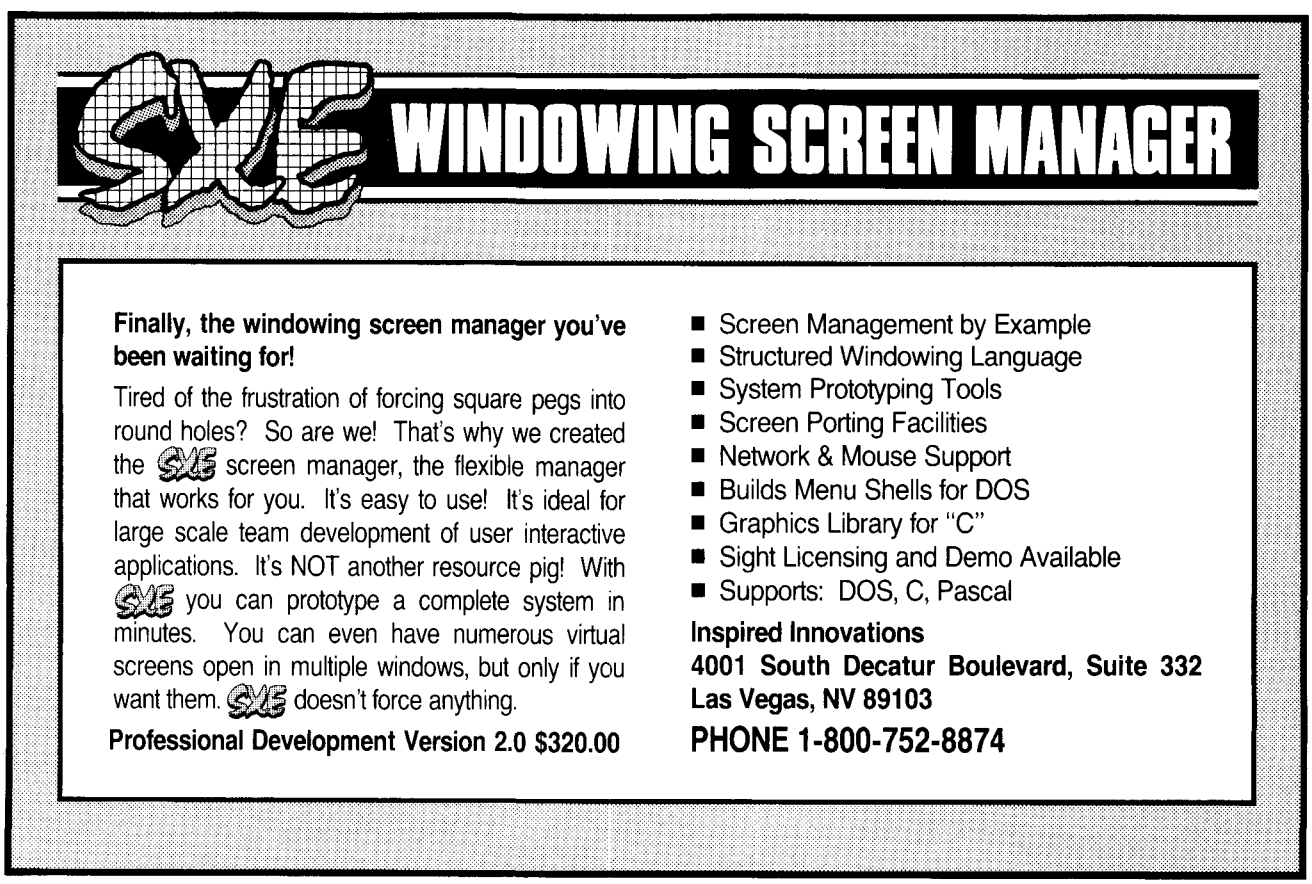
Ben Myers
Spirit of Performance, Inc.
Harvard, Massachusetts

Michael Abrash responds: *Good insights on your part, Ben, as usual. The thing about SuperVGA 800x600 vs. SuperVGA 1024x768 is that while the 800x600 setup code differs, the bitmaps are all the same, so you only have to write one set of graphics primitives. VESA may succeed in coming up with a standard 1024x768 mode set (although I think they're a little late there, too) but the bitmaps are* not *all the same: some are banked and some are segmented, and the banking architectures vary. This requires multiple implementations of graphics primitives, which for my money is worse than multiple mode-set implementations. And don't even speak of high-resolution 256-color modes!*

Yes, I'm living in Vermont; I work for Metagraphics, with an assist from the phone company and the airlines. By the way, MetaWindows supports the multitude of VGA implementations you've described; it is indeed a mess, and I wouldn't want to have to do it if it weren't my primary business. But that's why there are third-party libraries, eh?

I'd be interested in seeing your VGA benchmarks, both results and code, if possible. It's not clear to me how a write mode three screen-fill benchmark would differ from write mode zero, except for clipping at the edges—but I'd like to learn. I'm not surprised that the fastest VGA is on a motherboard; an AT-compatible bus can only run so fast (about 375ns per access), but a motherboard chip can be configured to run as fast as silicon will allow. Look for more fast motherboard VGSs in the future. PJ

We at *Programmer's Journal* like to hear from our readers. Please address correspondence to Editor, *Programmer's Journal*, 150 N. 4th Street, Springfield, OR 97477. You can also contact us via fax (503) 746-0071, MCI Mail (user ID 336-4248), or Compuserve (user ID 76676,427). All letters become the property of *Programmer's Journal* and may be edited for publication.


SVE
WINDOWING SCREEN MANAGER
Finally, the windowing screen manager you've been waiting for!
Tired of the frustration of forcing square pegs into round holes? So are we! That's why we created the SVE screen manager, the flexible manager that works for you. It's easy to use! It's ideal for large scale team development of user interactive applications. It's NOT another resource pig! With SVE you can prototype a complete system in minutes. You can even have numerous virtual screens open in multiple windows, but only if you want them. SVE doesn't force anything.
Professional Development Version 2.0 $320.00
Screen Management by Example
Structured Windowing Language
System Prototyping Tools
Screen Porting Facilities
Network & Mouse Support
Builds Menu Shells for DOS
Graphics Library for "C"
Sight Licensing and Demo Available
Supports: DOS, C, Pascal
Inspired Innovations
4001 South Decatur Boulevard, Suite 332
Las Vegas, NV 89103
PHONE 1-800-752-8874

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Redirections

By Frank D. Greco

Even though the first major hypermedia conference was held at the University of North Carolina, home of one of college basketball's biggest powerhouse teams, no one mentioned object-oriented databases. Unfortunately, the all-sports cable network, ESPN, didn't cover the event—much to my chagrin. When I called them, they said that ESPN refused to televise the event because with (the questionable) Jim Valvano as UNC Tar Heel coach, they couldn't possibly be objective... [pause for laughter to subside].

Seriously, it appears to me that a truly object-oriented database is essential for a powerful hypermedia system. And I'm not talking about object-oriented-like veneers around a relational database, which would be too slow. The very nature of relational databases just doesn't seem to map well to the structure of truly object-oriented architectures. Simulating objects in a relational database would entail multi-multi joins or cascading triggers (for referential integrity) or both, bringing a relational database to its knees in a hurry.

An object-oriented database can implement objects in a much more natural and efficient manner than a relational database. Although there are three or four commercial O-O databases currently available, their success hinges on a standardized query language. The current standard query language, SQL, is not culturally suited for the upcoming wave of object-oriented software.

While very powerful and extremely useful, SQL is inherently rooted in the database philosophy of relations, joins, selections, and projections. This framework is something quite alien to an object-oriented database. So what might the interface language of the near future be, you ask? Who knows.

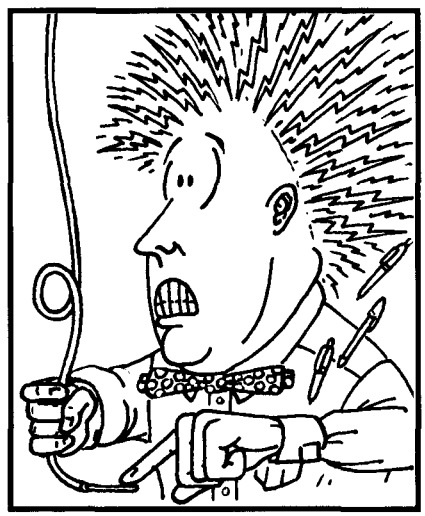


ILLUSTRATION BY JIM CARPENTER

COOL DEE RAM FABBED

The name may sound like the latest rap group from that alternate reality known as MTV, but it's actually a 64-megabit DRAM created by Hitachi researchers in their Japanese labs. That's 64 million bits, folks—64 times the addressability of your garden variety PC; 8 megabytes in 9 chips, almost the size of the original hard disk in the IBM XT.

What are we going to do with all this memory, you say? Well, with more modern (production-quality) operating systems like Unix System V.4 using some new virtual memory management techniques (borrowed from SunOS 4.0 and 4.1, and even from Carnegie Mellon's newcomer Mach), you might be able to memory-map large databases or files into main memory and benefit from the tremendous speedup. It's quite similar to using a RAM disk, but much more sophisticated.

Software that accesses data resident on a RAM disk still has to wind its way through the operating system's file system management code. Files that are memory-mapped don't have to be accessed that way because of a little trick that utilizes the DMA hardware more efficiently. Once the data (files or databases) is mapped into main memory, you access it as ordinary internal storage.

I recently had a chance to use this. One of my clients needed to display very large files in a scrollable window. The normal load time for the files from the hard disk was significant. Users soon grew weary of waiting. By using memory-mapping, I was able to "map in" a 22-megabyte file into a scrollable X window in a split second.

UNIX BASHERS

And puh-leeze don't give me that bull about Unix being an old operating system from the 60s. Microsoft has been trying to use that line as a marketing tool to support OS/2: "Unix is for engineers; it's so unfriendly," says Mean Green. So why did Bill Gates invest heavily in SCO? Was it a hedge?

Geez, they call OS/2 a modern OS when the most advanced machine-to-machine communication model is a "Named Pipe"? Gimme a break. Touting named pipes as a breakthrough is somewhat of an exaggeration, to say the least. When Steve Jobs was still Mr. Apple, he was once quoted as saying that "Unix is an old operating system for engineers."

This, of course, was in the old-Macintosh, pre-Sculley era, before Jobs was involved with Mach, which is (on the surface, at least) Unix. Mach presents the BSD Unix system call interface to programmers, but internally it's much different. Given its age, it should take a couple of years for Mach to mature, if it has the chance to.

WOW!

Two researchers from Phillips Research Laboratories in The Netherlands have used a minuscule tungsten needle to create the tiny pits in an optical disk. What's so special about this? The silicon the researchers used in the disk creates these "bits" in an area 10,000

Frank D. Greco is an independent consultant who provides industry analysis, systems consulting, and technological forecasting for Fortune 500 companies. Interested readers may contact Frank in care of Crossroads Technologies, P.O. Box 530, Fanwood, New Jersey 07023, (201) 754-7820.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

This team created your next generation editor. Tell us if you agree.

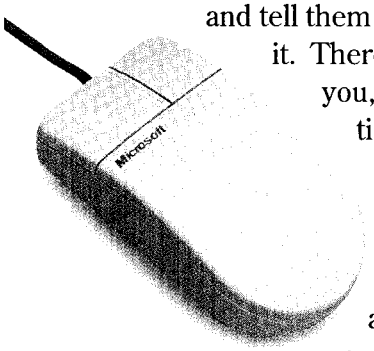
From left to right: ***Jim Besemer****, Project Manager, Specification & Design, User Interface;* ***Skip Russell****, System Primitives, Function Library, Error Recovery, Operating System Interface, Configuration and Setup:* ***Pat Thompson****, Extension Language/Compiler and Debugger:* ***Robynne Major-Jones****,* ***Howard Mercier*** *(not shown) Quality Assurance and Testing:* ***Eric Johnson****, Editor Core, Help Facility;* ***Milo Wadlin****, User Documentation.*

Announcing the Sage Professional Editor

For two years this team worked in secret to create the world's best editor. We believe they succeeded magnificently. Now we'd like to know what you think.

New development environments and the size and complexity of projects make your 80's editor inefficient and obsolete. So we built you an editor for the 90's.

Try the Sage Professional Editor for thirty days with no risk. Pound on it. Compare it to any editor you've ever used. Then call our team at 1-800-547-4000 and tell them how you like it. There's no risk to you, no obligation. Not much risk for us either. We won't be able to pry it away from you.

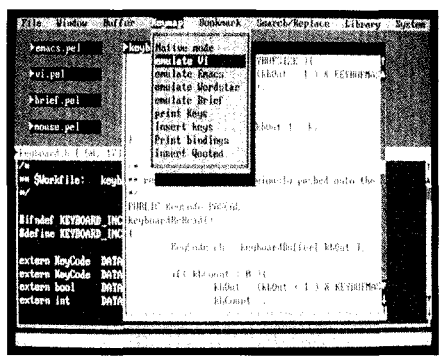


Works with or without a mouse.
Packaged with or without a Microsoft® Mouse.

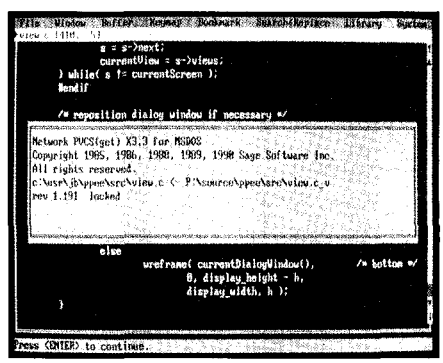
Right out of the box you'll be more productive with our editor than any you use today. The instant installation, elegant mouse support, advanced user interface, and point-and-shoot help get you running immediately. If you prefer the commands and keystrokes of a popular editor, our turnkey emulations duplicate them precisely, and you still gain the Sage Professional Editor's advanced features, windowing capabilities and powerful engine.

Later you'll make this editor truly yours by configuring the interface as you prefer. Every feature can be turned on or off as you like - from a clean screen, to tiled windows, to overlapping windows in various colors, pull-down menus, rulers, scroll bars, and line numbers - choose any or all and place them as you like.

Use the editor with or without a mouse - all functions are available without lifting your fingers from the keyboard. But the click-select windows, scroll bars, zoom, shrink-to-icon, block text selects, and other speedy mouse actions make editing extremely efficient.



Make our interface what you prefer, from clean screen to multi-window with drop-down menus and icons.



The editor environment provides seamless integration to the Polytron Version Control System (PVCS) or any other tool you care to connect.

Sage Software, Inc., 1700 NW 167th Place, Beaverton, OR 97006, (503) 645-1150, Fax (503) 645-4576, Int'l. (301) 230-3307, Int'l Fax (301) 984-3047.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Pop open the DOS window and the editor shrinks to just 4k. So you can back-task to compilers and other tools without leaving the editor.

This package is stuffed with value. It includes MS-DOS, OS/2 and Dual Mode versions on both 5 1/4" and 3 1/2" disks, templates for popular languages, and you can buy it with or without a bundled Microsoft® Mouse.

But the reason you will become addicted to the Sage Professional Editor is the power behind that pretty face. We looked under the hood of other editors and found wimpy editing engines dressed up with all sorts of bells and whistles. Unacceptable. We wanted an industrial strength engine. So we started with a powerhouse virtual memory system that allows us to edit huge files (up to 100MEG) in as many as 256 windows - over two billion lines. It makes maximum

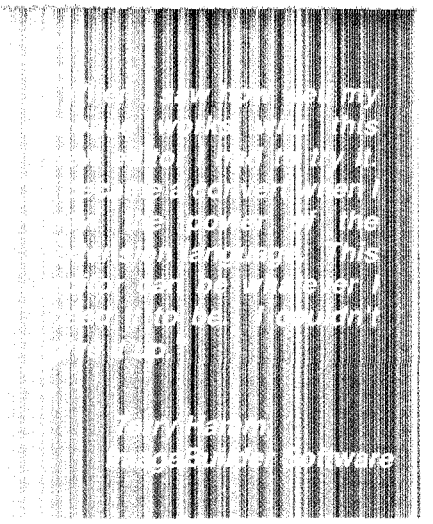
TurnKey Emulation Of

Vi | Brief | Wordstar | Epsilon/EMACS | Others

use of available memory (including EMS) and uses advanced processes to safely minimize file I/O. All higher level services use this powerful VM scheme. Consequently, there are no size constraints on the macro library and no limit to Undo/Redo. You can have 1000 bookmarks, anchors and saved positions per buffer.

Sage Software, Inc. has acquired Polytron, the company that brought you PVCS and PolyMake. Now this group proudly introduces their Professional Editor under the Sage banner.

DEVELOPMENT POLYTRON TOOLS



And then there's the extension language. The Sage Professional Editor uses a C-like extension language (100 percent PolyAWK compatible) and compiler/debugger that programmers find immediately intuitive. The function library gives access to nearly everything. From mouse click to menu, from word processing to external hooks. You can build the environment you need or want with the editor as the front end to your favorite tools. The seamless integration of the Polytron Version Control System (PVCS) is a sterling example of how cleanly you can hook external programs.

Emulations of Vi, Brief, EMACS, and WordStar, were written with the extension language. The source code for each is included. If you are addicted to some obscure editor you can modify a copy of emulator source and quickly make your own. Source for the menuing system, mouse interface, help and many other services is also included, so you can deeply modify the editor to be exactly what you want. Brief users who have a library of macros can use our translator to recompile them into our more useful and powerful C-like extension language.

It's easy to make the Sage Professional Editor a standard for your team or even the entire company:

1. Turn-key emulation means every programmer can use it immediately. Macros written to aid users apply to all emulations.

2. Provides a framework for integrating all your development tools into a single, consistent operating environment.

3. MS-DOS, OS/2 & Dual Mode versions are all included in the same box so *your* team can move into the 90's with the same editor.

4. Licenses are available for you, your team, division or company. The initial cost is low, and you continue to save as you add users.

MS-DOS | OS/2 | Dual Mode — All in One Package

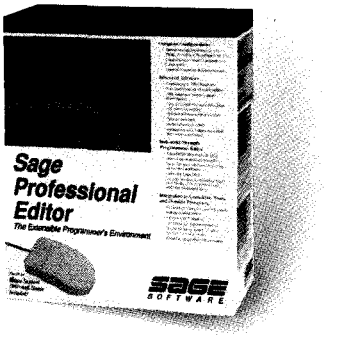
$295 U. S. List Prices. Single User Version.

$395 Includes a Microsoft® Mouse.

Specify serial, bus or PS/2 mouse. A 5-User License (without mice) is only $1,032.50. Call for prices on larger licenses. **Visa, MC** & P.O. orders:

1-800-547-4000

30 Day Money Back Guarantee



SAGE SOFTWARE

Circle 26 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All the windows had a large Batman logo right in the middle of the label!

times smaller than the current size of a CD pit.

Phillips Laboratories is projecting that in 5 years, CD disks using this technology will have 5,000 gigabytes (5 terabytes) of capacity—that's the equivalent of about 21 days of digitized audio, or 5 days of uncompressed video images at 24 frames per second. Mind boggling.

NEC is one of the few companies that has demonstrated devices capable of digitizing 3-D objects within 1/30 of a second by using special color cameras to measure the wavelength of diffracted xenon lamp light off the object. Combine this technology with devices that can create 3-D objects using digital data from a CAD/CAM program and you're onto something.

Professional software made affordable.

Sandpaper's System Debugger

The power of the 386 is yours for the taking.
System Debugger uses the 386 to create the ideal DOS development environment.
It uses the 386's protection, page tables and debug registers to provide the power of an expensive In-Circuit Emulator at a fraction of the price.

With System Debugger you'll ...
Set breakpoints on memory location and range reads and/or writes and **discover.**
⇒*When is the InDos flag incremented?*
⇒*What subroutine is overwriting my code?*
⇒*When does my malloc() space get corrupted?*
⇒*Am I overwriting DOS?*
Set breakpoints in ROM and **learn.**
⇒*Why does my software work in RAM but not in ROM?*
⇒*How does the BIOS work?*
Set breakpoints on I/O ports and interrupts and **know.**
⇒*Is a TSR conflicting with my software?*

Order now at no risk.
Money-back guarantee.

$99

Here is just a sampling of its features:
- Has symbolic debugging with source code of programs, Device Drivers, TSRs and ISRs.
- Runs standalone as well as in tandem with your current debugger.
- Doesn't slow down the system: load it by autoexec and it's ready when you need it.
- Stays active even if your program is hung with interrupts disabled.
- Displays undocumented DOS structures and memory map.
- Automatically comments interrupt calls so you don't need to look them up.
- Has an easy to learn intuitive windowed user interface.
- Finds those elusive annoying bugs which only exist in real time.
- Runs under DOS on 386SX/386/486 AT compatibles with 256K extended memory.
- Runs entirely in extended memory so that 640K is left where you need it.

To order, Call or FAX (818) 705-2657, or send a check to:

Sandpaper Software

P.O.Box 18935, Encino, CA 91416-8935

Please specify diskette format. CA residents please add sales tax.

GET REAL

I recall seeing a product at Comdex produced by Roland (yet another multi-talented Japanese company) that produced 3-D gear prototypes by melting layer upon layer of wax from a block via a computer-controlled laser. Competing technologies use still other methods for creating objects.

Just imagine the field day that people in the entertainment industry are going to have with this one. You'll be able to walk up to a machine, plug in your optically-read credit card that holds 10 megabytes, and download a description of your whatever. VCR tapes will come with a card so you can create a model of the leading actress by taking your optical card to the local print—er—realization shop. Imagine getting an optical card when you go to the movies and taking it to your favorite fast food outlet to generate a little teenage mutant ninja turtle model. Or think about digitizing your teeth at your next dental appointment to make instant caps or false teeth.

Hmmm...Is anyone out there working on a PostScript-like language for real 3-D rendering?

KEEP IT SIMPLE, SPARC-Y

As opposed to CISC processors, which have in excess of one million transistors, most SPARC implementations have approximately 55,000. This is similar to the Cray architecture (lots of registers, large cache, and highly pipelined).

Matsushita, primary investor in Solbourne, appears to be more than just a little interested in producing low-cost SPARC-based machines for sale in Japan as well as in the U.S. and Europe. The two companies plan to produce a

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

10-MIPS workstation for under $8000 in late 1990.

Speaking of Solbourne, their OI X Window C++ interface library supports both Open Look and Motif. I was given a demo at the last X conference and was thoroughly impressed. Their window manager, gwm, was also impressive. At the click of a menu, all the windows changed from a Motif look-and-feel to an Open Look look-and-feel. Then another menu selection changed the windows from Open Look to a Batman look-and-feel; all the windows had a large Batman logo right in the middle of the label! Yet another menu selection made the window frames look like wood grain.

Sun has quietly upgraded their 12.5 MIPS SPARCstation I (whose formal name is the Sun 4/60) to use a SPARC processor with a higher clock rate. All SPARCstations now shipped are rated at 15.8 MIPS. That's 3 extra MIPS for no additional cost. Now that the SS1 (SPARCstation I) is one year old, it seems like it's time for a new member of the family. Moving the clock rate up gives Sun more headroom to slip in a slightly slower but cheaper machine at the low end.

At the last Sun User Group meeting in Anaheim, Andrew Bechtolsheim, the originator of the Stanford University Network (SUN) and one of the founders of Sun, alluded to the rumored SPARCstation II family as both smaller and denser. The Sun hardware guru said that Sun has already shrunk the size of the SS1's motherboard down to 1/4 the size of the original in their labs.

LOOK INTO MY CRYSTAL BALL

If I was allowed to conjecture (but of course, *mon ami*), I could see low-end, end-user 10-MIPS SPARC workstations listing for about $4000 to $5000 and higher performance 40-50 MIPS developers/graphics workstations going for around $15,000 to $30,000. The current SS1 would then be lowered in price so that a fully-loaded color system with the GX graphics accelerator would be in the neighborhood of $10,000 to $12,000. Now this is just my own rambling, of course, and could be furthest from the truth, but it certainly sounds nice, doesn't it?

Cypress Semiconductor is in the process of putting together a superscalar implementation of the SPARC microprocessor which will be capable of executing two instructions per cycle, yielding 80 MIPS performance. LSI Logic and Hyundai Electronics are also developing an 80 MIPS SPARC chip for incorporation into a future Hyundai workstation, which will be announced in early 1991. This implementation would also be superscalar (multi-instruction per cycle).

Without a doubt, the SPARC will be one of the dominant microprocessor architectures over the next 3-5 years; its use will be much more widespread than use of the 68030/040 and the 80486. In case you hadn't noticed, SPARC is already out-selling the MIPS RISC chips by a factor of ten to one.

SCROLLING THRU THE PARC

Rumor has it that Open Look's "shine mark," used to iconify a window, was created by Sun Microsytem's VP-of-whatever-he-wants, Bill Joy. The story goes that he was playing around with a paint package and by accident XOR-ed some pixels.

Rumor also has it that Sun has a royalty-free license from Xerox for their user interface technology because Bill Joy wanted to give credit where credit was due. Xerox gets access to the SPARC license in return.

Toshiba Corporation has announced that they have developed a font-rasterizer chip that will generate Bezier curves using a dedicated hardware divider. This chip produces four dots per clock cycle. A version of this chip running at only two MHz would produce a full page bitmap from the Bezier curve description in only one second. Only the fastest printing engines could keep up with this chip.

OPTIMIZE, MY GOODNESS!

Unix optimized for SPARC? Sure! Of course Sun's Unix is optimized for the SPARC processor. All versions of Unix are (or should be) tuned for the native processor running the OS. It's the nature of such a remarkably portable piece of software. Don't you think that when OS/2 (yeah...whenever) is ported to

PAINT POWERFUL DATA-ENTRY SCREENS WITH POWER SCREEN™

Design screens and menus with PAINT™, placing input fields and messages precisely where you want them to appear in your final application. PAINT stores screen definitions in a single file — which can be modified without recompiling or relinking your application. Your program manages screen display and I/O through the POWER SCREEN Runtime Library. All essential features are supported, including:

◆ Block multi-field or single-field I/O ◆ Automatic range checking ◆ Configurable editing/menu key definitions ◆ Application context-sensitive, cross-referenced help ◆ Virtual/automatic scrolling screens within viewports ◆ Multiple, overlapping viewport display ◆ Plus much, much more!

The Runtime Library can be linked directly with Turbo Pascal applications, or installed as memory resident.

All this for just $149.00

Includes complete sample programs, a comprehensive reference manual, and the Norton Instant Access program and guides to assist you during program development. And, we offer a 30-day money back guarantee. Supports Microsoft C/QuickC, Turbo C, Turbo Pascal, QuickPascal and QuickBASIC.

Other powerful products from Blaise Computing:

C ASYNCH MANAGER™	**$189**
ASYNCH PLUS™	**$189**
View232™	**$189**
C TOOLS PLUS™	**$149**
Turbo C TOOLS™	**$149**
POWER TOOLS PLUS™	**$149**
POWER SEARCH™	**$149**

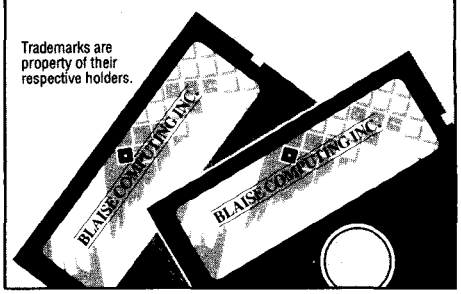
Call today for more information

(800) 333-8087 Fax (415) 540-1938

BLAISE COMPUTING INC.

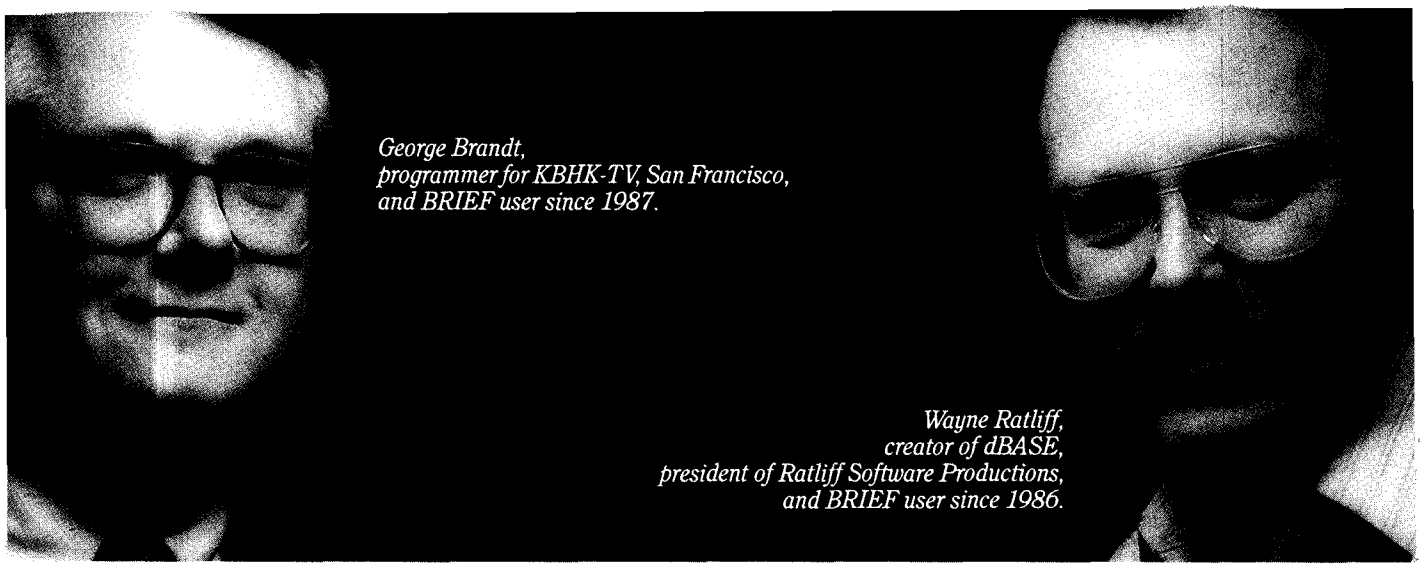
2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

Trademarks are property of their respective holders.



Circle 14 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



George Brandt,
programmer for KBHK-TV, San Francisco,
and BRIEF user since 1987.

Wayne Ratliff,
creator of dBASE,
president of Ratliff Software Productions,
and BRIEF user since 1986.

Of course they need completely different kinds of editors. That's why they both use BRIEF.

Let's face it. There's no such thing as two programmers who think alike. Or who work alike.

Because everybody's got their own style, their own habits. Their own quirky little requirements. As well they should. So, where are you going to find one editor that satisfies everyone? Simple. Look inside a BRIEF® box.

It's the right editor, right out of the box.

What you'll find inside is an enormously powerful, astonishingly easy, and highly-evolved program editor. One that works at blinding speed. And that's loaded with features designed to help you be more productive. One guaranteed to give you excellent results within one short hour of booting up the first time.

You'll find features like automatic indenting, infinite windowing, and powerful regular expression searching that goes well beyond the ordinary text-matching you're probably used to.

You'll also find the features that made BRIEF famous. Including our near-miraculous UNDO, that doesn't just un-delete your last keystroke. It actually reverses the effect of any command. So you can explore, experiment, even make gigantic mistakes with hideous consequences, and just "undo" them. And then continue to undo, backing-up through 300 steps.

What you want is what you get.

What you'll also find is an editor that delights in adapting itself to you. With the simple, menu-driven SETUP program, you can quickly customize indenting style, support for any of 36 compilers, keyboard configuration, rate of keyboard auto-repeat, and much more.

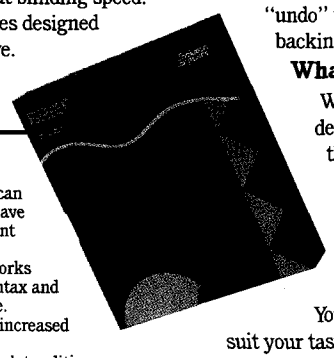
You can even edit BRIEF's macros to suit your taste. Or write your own in a snap.

Computer Language said that BRIEF is capable of "almost complete customization." And with new release 3.0, BRIEF is even more powerful and flexible.

For every programmer, it's a different BRIEF.

At last count, there were over 52,000 copies of BRIEF in the field, making life easier and working time more productive for all kinds of programmers—working in all kinds of languages. Programmers as different as Wayne Ratliff is from George Brandt. Which means there are over 52,000 different ways of working, and BRIEF fits them all.

Why not get BRIEF going to work for you? There's no risk. In fact, if you're not completely satisfied that BRIEF is exactly the editor you've been looking for, just return it within 30 days, and we'll return your money. No questions asked.



New BRIEF 3.0 highlights

- With our new C-like macro language, CBRIEF, C programmers can write macros on the fly and not have to switch between C and a different macro language.
- The new source level debugger works with both the original BRIEF syntax and the new CBRIEF macro language.
- Maximum lines per file has been increased to 4 billion.
- We've added to and improved template editing and smart indenting for ADA, Cobol, BASIC, FORTRAN, Modula-2, Pascal, and C.
- We've added editable, multiple keystroke macros that can be saved and restored.
- You can even save and restore your window set-ups.

Plus the features that made BRIEF famous.

- 300-level UNDO safety net
- Flexible windowing, unlimited quantity
- Unlimited number of files
- File size limited only by disk space
- Regular expression search and replace
- BRIEF's LISP-like macro language
- Toll-free tech support

International Distributors: England (0763) 73457 West Germany (069) 34 72 26
France (1) 46 22 9988 Australia (03) 663 6580 Denmark (42) 88 7249 Norway 03-89 22 40
Austria (1) 31 02031 Belgium (3) 829 0581 Israel 4 238757 Sweden (0) 300 70910
Switzerland (31) 586655

All product names are trademarks or registered trademarks of their respective owners. © 1989 Solution Systems

BRIEF 3.0
The Programmer's Editor

Solution Systems™
541 Main St., Suite 410
South Weymouth, MA 02190
FAX 617-337-7719

Try BRIEF at our risk. $199 plus $6 shipping (U.S. and Canada). 30 day money back guarantee.

Name _____
Company _____
Address _____
City _____ State _____ Zip _____
Phone _____

Paying by ☐ Check ☐ MC ☐ VISA
Card # _____ Exp. date _____
Signature _____
Disk size ☐ 5-1/4″ ☐ 3-1/2″ Format ☐ DOS ☐ OS/2

Mail this coupon or call toll-free today.

800-821-2492
Extension 496
Massachusetts & outside U.S. call 617-337-6963

Circle **47** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

non-80X86 architectures, it won't be optimized for its host processor? So, OSF's bleating was just a cover-up to cause more market confusion about Unix. This gave the traditional Unix-bashers, such as IBM and DEC, a chance to gather their thoughts on how to handle the growth of Unix.

How about someone designing an operating system that *learns* how to optimize its I/O and process/thread creation strategies?

MINUET AND ROCK-N-ROLL

Isn't it funny that Apple is now targeting their advertising toward corporate MIS-types, while IBM's new advertising is designed to appeal to yuppies? It's a wacky world.

IN FOR THE SHORT HAUL

NEC's PASOLINK is a point-to-point microwave radio transmitter/receiver that allows you to connect multiple local area networks through the air for a maximum of about 12 miles (20 kilometers for those who don't know that the slogan "think metric" is about 25 years old). Transmission band width is in the neighborhood of 2 to 44 megabits per second, wide enough to send digital, voice, or video data to another close facility at speeds surpassing Ethernet. Because all this is done through the air, no special tower is required. Neat stuff.

GOODBYE GRAINY DAZE

The highest resolution gray-scale monitor around is MegaScan UHR-2007. This monitor is 2560 X 2048 X 8-bit, with a physical resolution of 200 dpi. MegaScan also makes the highest resolution b/w monitor, UHR-3000, which is 4096 X 3300 and a real 300 dpi monitor.

The innovative Sharp Electronics company has shown a flat 14-inch color LCD display that has VGA resolution and can respond at video rates. With this technology in hand, the Sharp-minds people are strongly rumored to be preparing their announcement of the ever-elusive flat-TV in early 1991.

A true color hologram, viewable in normal lighting conditions, was shown at a recent Display Holography symposium by a graduate student from Oxford University. This amazing device provided true color rendition from all viewing angles. Now all we need is a real-time, computer-generated version that can display the image in 3-D space. Apparently TI has shown a prototype of such a beast in their labs.

Sun, Compaq, et al. are part of a consortium to develop 17-inch color LCDs. Does this mean laptop workstations? It's only a matter of time. Toshiba, who has demonstrated an impressively sharp color LCD display for their ichi-ban laptops, is all set to announce their low-end SPARC-based workstations in Japan within a few months (and maybe even sooner). According to my contacts from Japan, it will be a low-end workstation, probably around 10 MIPS, running SunOS (with System V.4 in the future).

Targeted price, you ask? Try $6,000 with a 200 megabyte drive—that's what Toshiba is considering. Also look for Toshiba to announce a higher density hard disk for their speedy T5200 luggable; they're apparently in beta mode for a 300+ shock-resistant internal drive. They also have plans to support the 33 MHz 386 with an update to the T5200.

I really get a kick out of writing X Window software (X11R3) using a full-blown System V.3 Unix on my Toshiba T5200 portable, all on my kitchen table. It's funny, though—I once thought that 100 megabytes would be sufficient storage. Silly me.

PERFECTLY PICKED POSIX

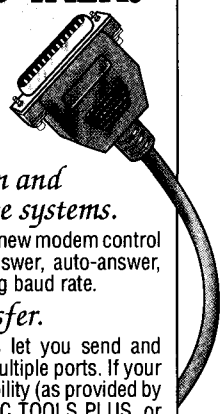
Microsoft's Xenix director Paul Sribhibhadh (if you say his name backwards, you can hear the phrase "polling is dead") said in a recent *Infoworld* interview that OS/2 would be Posix-compliant in late 1989. Now didn't I say that in the Vol. 5 No. 1 issue of *PJ* way back in 1987?

MANZI'S MEGALOPOLIS

The recent merger between Lotus and Novell bodes well for Sun and Sybase. Remember that Lotus owns 15% of Sybase, so you can bet money that Sybase's network interface library will

WE HAVE WAYS TO MAKE YOU TALK.

(And let you listen in, too)



with tools from Blaise Computing.

Talk to your modem and connect with remote systems.

Go online with ease with our new modem control routines. Initialize, dial, answer, auto-answer, and auto-adjust for incoming baud rate.

Let's talk file transfer.

XModem/YModem routines let you send and receive multiple files over multiple ports. If your program has ISR/TSR capability (as provided by our POWER TOOLS PLUS, C TOOLS PLUS, or Turbo C TOOLS), you can talk file transfer in the background while your foreground process talks data input, or database query, or whatever.

The file transfer capabilities include 1K packets, CRC error checking, YModem (multi-file transfers with file name and size preserved), automatic adaptation to incoming packet size and error detection method. And we're talking fast — all file transfer routines have been optimized for maximum throughput.

Let's talk basics.

C ASYNCH MANAGER™ and **ASYNCH PLUS™** retain the features which have made them the libraries of choice for asynch projects — buffered interrupt-driven input and output to multiple COM ports with speeds up to 19200 Baud, XON/XOFF protocol, hardware handshaking and much more.

C ASYNCH MANAGER supports Microsoft C, QuickC and Turbo C. ASYNCH PLUS supports Turbo Pascal and QuickPascal. Both products require DOS 2.00 or later and an IBM PC, XT, AT or PS/2 machine or true compatible.

Let's talk price.

Why pay more? Our ASYNCH packages are just $189, including complete source code, a comprehensive reference manual with extensive examples, sample programs and online help.

Turn that serial cable into a party line with View232!

Debugging serial communications doesn't have to leave you talking to yourself. **View232™** turns your computer into a serial data analyzer that lets you listen in on the conversation between any two serial devices — and this party line provides a transcript of what was said!

View232 displays the data as it flows in both directions. Save a whole transmission in a buffer, then browse through or search the buffer for a pattern or specific character. And you can save the data to disk or print it for later study.

VIEW232 is easy to use. And we supply the cable, all for just $189!

Let's talk track record.

Blaise Computing has produced a collection of tools that are unsurpassed for reliability, flexibility, and ease of use.

And, we're talking guaranteed.

If during the first 30 days you're not completely satisfied, we'll refund your money.

Call (800) 333-8087 today!

BLAISE COMPUTING INC.

2560 Ninth Street, Suite 316
Berkeley, CA 94710
(415) 540-5441
FAX (415) 540-1938

Trademarks are property of their respective holders.

Circle 2 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

soon be ported to Novell's LAN protocol. And it will only be a matter of time before Novell implements Sun's Network File System (NFS) on their networks, subsequently linking all of the PC's, Macintoshes, Suns, and mainframes in your entire corporation.

Imagine using Sybase's Open Server library to write a "gateway" to other non-Sybase databases (for example, DB2, ISAM) over Novell. The new Sybase DataLens (SQL) interface into Lotus would be your interface to that Open Server. You could then have access to your corporate databases via an identical Lotus spreadsheet interface, whether on your PC's, Macintoshes, or SPARCstations.

Speaking of ever-hot Sybase, apparently their public offering is imminent. Sybase has been using the lure of stock options to potential employees to bring top-notch talent to the company. I would guess the IPO would be in mid- to late 1990, depending on market conditions.

MY X WAS A HEAVY WEIGHT

In response to a question about when to implement X Window's windowless, lightweight "gadgets" vs. heavy-weight "widgets," Oliver Jones, frequent X Window conference speaker and author of a popular X Window book, told his audience at a recent X conference in Boston that it's too early to attempt to select one over the other based on performance. He stated that it would take many years for the X Window system to stabilize and for the bugs to be shaken out.

I also remember Sun Microsystem's window guru, David Rosenthal, mentioning that he had found pieces of code in the X Window server which were subject to race conditions. Knowing this, you can understand my fear when I

LISTING 1

```
#include <stdio.h>
#include <dos.h>
#include <ctype.h>

#define GRAPH_LENGTH    50
#define BLOCK           219

#define CHAR_HORIZLINE  196
#define CHAR_TEETOP     194
#define CHAR_TEEBOTTOM  193
#define CHAR_ENDTOP     191
#define CHAR_ENDBOTTOM  217
#define CHAR_VERTLINE   179

char *commaize();
static char _desc_[ ] =
"disk.exe - Determines disk capacity - Copyright (c) 1990, Frank D. Greco";

main(argc,argv)
char **argv;
{
     struct diskfree_t ds;
     unsigned drive;
     unsigned long total;
     unsigned long bytes_per_cluster;
     float glen;
     char *lptr;

     if (argc == 1)
         _dos_getdrive(&drive);
     else
         drive = tolower(*argv[1]) - 'a' + 1;

     if (dos_getdiskfree(drive, &ds)
         fprintf(stderr, "Invalid Drive.\n"), exit(1);

     printf("              Drive:  %5c\n", 'A' + drive - 1);
     printf("    Total Clusters:  %5s\n", commaize((long)ds.total_clusters));
     printf(" Available Clusters:  %5s\n", commaize((long)ds.avail_clusters));
     printf("    Percentage Full:  %5.2f %%\n",
            glen((float)(ds.total_clusters-ds.avail_clusters)/
            ds.total_clusters)*100.);

     graph((int)glen);
     printf("Sectors Per Cluster: %10s\n",
            commaize((long)ds.sectors_per_cluster));
     printf("  Bytes Per Sectors: %10s\n",
            commaize((long)ds.bytes_per_sector));

     bytes_per_cluster = total = ds.bytes_per_sector * ds.sectors_per_cluster;
     total *= ds.avail_clusters;

     printf("      Disk Capacity: %10s\n",
            commaize(ds.total_clusters*bytes_per_cluster));
     printf("               Used: %10s\n",
            commaize((ds.total_clusters-ds.avail_clusters)*bytes_per_cluster));
     printf("    Total Space Left: %10s\n", commaize(total));
 }


graph(len)
int len;
{
     register int i;
     int max;

     for (i = 0; i < GRAPH_LENGTH-1; i++)
         putchar( ((i+1) % 5) ? CHAR_HORIZLINE : CHAR_TEETOP);

     putchar(CHAR_ENDTOP); putchar('\n');

     max = len / (100 / GRAPH_LENGTH);

     for (i = 0; i < max; i++)
         putchar(BLOCK);

     for ( i = GRAPH_LENGTH-max; --i > 0; putchar(' '))
         ;

     putchar(CHAR_VERTLINE); putchar('\n');

     for (i = 0; i < GRAPH_LENGTH-1; i++)
         putchar( ((i+1) % 5) ? CHAR_HORIZLINE : CHAR_TEEBOTTOM);

     putchar(CHAR_ENDBOTTOM); putchar('\n');
 }


#define MAXNUM  64

char *commaize(num)     /* Put commas after every 3 digits */
long num;
{
     register int i = 0;
     static char buf[MAXNUM];
     register char *cp = buf + MAXNUM - 1;
     int digit;

     *cp-- = NULL;

     while(num >= 10)
     {
         i++;
         digit = num % 10;
         *cp-- = digit + '0';
         if ( i >= 3)
             i = 0, *cp-- = ',';
         num /= 10;
     }
     *cp-- = num + '0';
     return cp + 1;
}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

heard about one group of government software developers working on an air traffic control system using X Window. Scary thought, isn't it?

MAKE MINE BETTER

There's something about MAKE that I don't like. I mean, sure, I use it. And I wouldn't do without it but...there's gotta be a better way. Having a human describe a substantial MAKE file down to the last detail means that human is doing more work than necessary. MAKE is ripe for redesign (especially a graphical MAKE). There are some utilities that read your source and create the MAKE file for you (and some C compilers have an option that does this also). But that's only half of the process. The new-generation MAKEs, derived from databases (right on!), are steps in the right direction.

I have it from an inside source that there will never be *any* Xenix articles written for *Microsoft Systems Journal*. Apparently Bill Gates mandated that no Unix-related articles be included in *MSJ*. Much to their chagrin, Gates said this in front of a gathering of his top-notch Unix programmers. OS/2 vs. Unix—a war? Nahhh!

HUNGARIAN GOULASH

I'm sorry, but I think the "Hungarian Notation" (named after its originator, Microsoft's Charles Simonyi) used by Microsoft's Windows programmers is awful. It goes against two major programming language axioms: that identifiers be mnemonic and reflect their real-world representation. It also promotes programmer non-portability. Since everyone has a different concept of "Hungarian Notation," each version can be different. So, declarations like char **lpszCmd* and char **fpCmd* can both mean far pointers to char *. What's the difference, you say? A far pointer is a long pointer...that's the point.

Why concentrate on details like this? I have taught Microsoft Windows to experienced C programmers several times, and *every* time the Hungarian Notation gets in the way—so much so that I've given up on it. Embedding the data type into the variable name confuses the programmer and detracts from the application model. It reminds me of debugging C++ programs on a Unix system using Bjarne Strousup's early C++ preprocessor (which converts C++ to C). The C++ preprocessor converts C++ identifiers to C identifiers that have their C++ data types embedded in them. Have fun debugging that!

A LITTLE UTILITY

Just so you software junkies out there could satisfy your craving, I've included a little ditty (DISK.C) that displays disk capacity of any given disk in a somewhat graphical form (see Listing 1). The sample output is below. PJ

FIGURE 1

```
Drive:                  C
Total Clusters:         16,324
Available Clusters:     5,711
Percentage Full:        65,01%
```

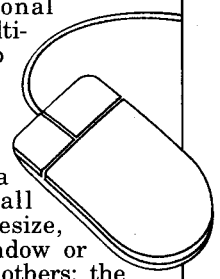
```
Sectors Per Cluster:    4
Bytes Per Sectors:      512
Disk Capacity:          33,431,552
Used:                   21,735,424
Total Space Left:       11,696,128
```


6.0 AND COUNTING!

Integrate sophisticated features into your Microsoft C and QuickC applications with

C TOOLS PLUS/6.0™

C TOOLS PLUS version 6.0 is filled with many advanced routines for developing high-powered C applications, including: virtual, stackable menus and windows with full mouse support and optional "drop shadows"; multiple virtual pop-up help screens; a miniature multi-line editor for gathering user responses in a robust fashion; a single function call which can move, resize, and promote a window or menu on top of all others; the ability to update covered windows automatically when they are written to; support for EGA, VGA, and MCGA text modes including 30-, 43-, and 50-line modes; support for the enhanced (101/102 key) keyboard.



All this and more for only $149!

C TOOLS PLUS/6.0 also contains functions for writing interrupt service routines; creating pop-up memory resident applications; general memory "peeks" and "pokes"; access to the DOS PRINT utility; as well as many other general utility functions and macros.

COMPLETE PROFESSIONAL PACKAGE.

Blaise Computing's function libraries offer easy to use solutions to your programming needs. You get source code, complete sample programs, and a comprehensive reference manual with extensive examples. Supports QuickC and Microsoft C 5.0 and later.

30 DAY GUARANTEE.

If during the first 30 days you are not completely satisfied, we'll refund your money.

Other powerful products from Blaise Computing

C ASYNCH MANAGER™ $189
ASYNCH PLUS™ $189
VIEW2.2™ $189
POWER SCREEN™ $149
Turbo C TOOLS™ $149
POWER TOOLS PLUS™ $149
POWER SEARCH™ $149

Call today for more information

(800) 333-3087

BLAISE COMPUTING INC.

Circle 15 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Deliberately Breaking The Rules

By Paul Barkley

Here in Washington, D.C., the Beltway is assaulted by rain and choked full with government employees headed home. In between the economy cars filled with GS grades and the unmarked, full-sized cars filled with hard men (and women) who work odd hours for agencies that don't even like to use their initials, Beltway Bandits in high-performance Porsches and BMWs zip through openings in the clogged traffic at breathtaking speeds approaching 20 miles per hour. It's a good day for German antilocking brakes, but it's not a good day for the Bandits. There are scandals afoot in our nation's capital, and the Bandits are feeling nervous.

I'm not talking about scandals involving selling arms to Iran, sleeping with the wrong people, or even (perhaps you remember this) breaking into someone's psychiatrist's office. Nah. That stuff happens all of the time around here.

I'm talking about serious stuff. I'm talking about doing something that screws up your consulting contract, something that might get you barred from all future work, or maybe even thrown in jail. I'm talking about highly successful, motivated entrepreneurs who have somewhere crossed the wavering line that separates the good guys from the bad guys. These driven individuals keep succeeding until the very practices that put them on top then put them away.

TRICKS OF THE TRADE

Admittedly, there are levels of wheeling and dealing, and maybe you have some trouble identifying with a guy who pays people in the Pentagon to give him the inside information on multimillion-dollar contracts. But let's look at some interesting tricks of the trade that you and I, as "normal-sized" consultants, may already be using.

Paul A. Barkley can be reached at 13312 Neerwinder Place, Germantown, Maryland 20874.

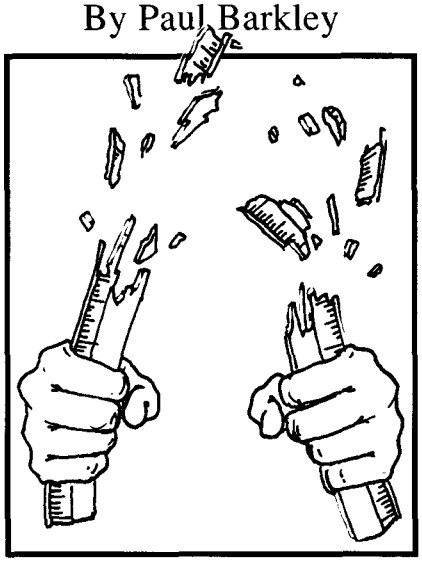


ILLUSTRATION BY JIM CARPENTER

Doing some of these things could help you triple your business. Maybe—or get you some free publicity on the front page of your hometown paper.

BUSINESS ETHICS

The topic of business ethics is pretty dry stuff, right? Let's skip the academic discussion about right and wrong in the abstract; Socrates just stepped out for a drink at the Hemlock Inn. Let's talk about some of the practices I've observed recently in consulting, and let you ponder whether they're right or wrong, or even practical. Remember, if you or any of your I.M. force are caught or captured, the Secretary (not the secretary) will disavow all knowledge of your actions. (I've always wanted to say that.)

I'll start with my favorite moral dilemma. Your client asks you to build a system that you don't think will work. Maybe you think the whole idea is unworkable, or perhaps you just think the client's choice of development language or operating system is not smart. Do you refuse to do the job?

This question generated an amazing amount of heat last year on a local BBS. The purists, of course, maintained that it is the consultant's duty at the least provocation to tell clients that they're full of beans (or other substances), and that only the consultant knows what's right. This attitude was generally supported by Forth programmers, SQL visionaries, and others who get frustrated when the world does not see the inherent correctness of their particular opinion. (I can see why the world may not agree with the opinions of zealots and fanatics, but I've never understood why everyone doesn't agree with my own enlightened views. Ahem.)

Anyway, these folks are convinced of the moral superiority of their views, and feel that the client is ill-served by advice other than their own. Simply put, they are adamant, inflexible, and unwilling to perform any services for a client that are not completely consistent with their own world views.

There's an advantage to this all-or-nothing mode of thinking, assuming that the technical position espoused has merit. If you refuse to compromise, you won't get sucked into some of the little client games that we're going to talk about in a minute. If this is your modus operandi, then you, like Lancelot, may have the strength of ten, for your heart is pure. You may also have a lot of time on your hands to enjoy your moral superiority, because chances are you won't have that much business.

THE OPPOSITE VIEW

Another breed of consultants, usually those with a keen eye for financial gain, believe that they were put on earth to give clients anything said clients want, for a price. If the client wants to automate the payroll for a two-person business, they'll be happy to build a $20,000 custom system, with annual maintenance charges of $4,000, for a job that could easily be done by hand.

The rationale is that clients know what they're willing to pay for, and if they are willing to back their wants with cold cash, then why should a consultant rock the boat? These consultants tend to be busy people, always building this system and that, and taking in a lot of

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

money. They also tend to wind up in court more often than other consultants. This attitude can attract a great deal of unpleasantness.

PROFESSIONAL OBLIGATIONS

I personally have always felt that it's my obligation as a professional to point out to my clients any problems with systems that they propose. I've been able to avoid every single disastrous system (so far) by simply saying, "Sure, I'll be happy to build that for you. It's going to cost you about $40,000, and might work OK. But before we spend your money, let me show you an alternative that you may not have considered. I think we can build the system this other way for about $10,000."

Most people are not complete morons. If you don't get in their face about what a dumb idea they have, you can let them decide on the basis of how much each idea is going to cost, and what the technical merits of each will be. This is a smooth way to reach an intelligent, mutually agreeable choice.

DECISIONS, DECISIONS

Let's look at a non-computer example. I went shopping for a large-screen TV the other day. I was totally bewildered by all the choices, even though I had dutifully read about half a dozen of the top performers in a consumer publication. The salesman showed me all sorts of different models with all kinds of different features. I was able to decide which features were important to me, and he was able to show me three or four that fit those specifications. Since there was a 100% price difference between the models, making the intelligent choice was easy.

While I was there, I overheard a different salesperson talking to a couple that was trying to decide whether to purchase a deluxe model that sat on an impressive pile of speakers. The speaker base added another $500 to the cost of the TV. The salesperson pointed out that the speakers sounded great, and that without the speaker base they'd have to put it on a sturdy table or purchase a TV stand for perhaps $200 anyway.

Note that the salesperson didn't tell them that only an idiot would buy the speakers, or for that matter only an idiot would buy the TV without the speakers. The point is, the salesperson presented some reasonable choices, and let the buyer decide.

In fact, if you think about the types of salespeople you've met, you'll notice that the good ones listen to what the customer wants, and then find reasonable choices based on what the customer is saying. They don't point out how much smarter they are than the customer. Well, successful salespeople don't. Ever get into an argument with a hot-shot car salesman? If so, did you subsequently buy a vehicle from that person? No way in Hades.

Now buying a car or a TV or a stereo is a little different from hiring a consultant/system developer. Although the technical merits of various consumer goods may be somewhat difficult to compare, and somewhat subjective as well, there are generally very few truly bad choices that can be made. If you listen to stereo equipment and pick something that you like, it's hard to go wrong. About the worst that a salesman can do is sell you speakers smaller than your amplifier (and of course that's pretty bad when you blow little pieces of paper all over the room). That error is fundamental, and easily avoided. If you're truly fanatic about your gear, you're smart enough to avoid more subtle problems as well.

MORE MISMATCHES

What about computer sales? We all know it's possible to purchase a mismatched set of computer hardware and software. Bad decisions, endorsed by an incompetent computer salesperson, are easily made. Half the hardware and software in the world doesn't work quite right when put together, even though these products are supposed to coexist nicely.

If you're in the business of selling off-the-shelf hardware and software solutions, you know that the stuff either works together or it doesn't—if you've ever tried that particular combination yourself. If you haven't, you've been burned enough by now not to go around making hasty promises. Odd combinations have a way of causing a lot of trouble. So if a client/customer asks for something that you know won't work, you'd be out of your mind to sell it to him. Why tell a customer that the new Warthog Writer will import dBASE III DBF files when it won't? You simply say that it won't work, and then you look for good alternatives.

Taking the last step in this line of reasoning brings us back to custom system development and our original question. If your clients want something that you know won't work, do you tell them?

I think that the same rules apply. If you know something won't work, you'd be out of your mind to not tell a client. All you're going to get is a lot of hassles (and bad karma or a one-way ticket to hell, if you believe in either of those things).

ONE BAD SITUATION

The problem, though, is that most custom system development is a lot more subtle than that. Based on your experience with similar systems, you need to be able to project how well something will perform, and whether it will meet the needs of the client.

I was involved in one bad situation where the hardware vendor sold an old client of mine one of those cheapo LANs, the ones that have no dedicated file server. It's possible that somewhere, someone is getting a lot of utility out of such an arrangement, but I personally didn't think it was all that great an idea. I went along with it, though, after stating, in writing, that I could not guarantee the performance of the custom application on that sort of hardware platform.

NOTHING BUT TROUBLE

I should have gone further. I should have said (gently but firmly) that if the client wanted to proceed with that alternative, I would not participate. As it turned out, it was nothing but trouble. The hardware vendor was dismissed after a ton of mutual nastiness, and a new vendor was hired.

I had a number of discussions with my client's new vendor, who was a Unix hotdog. His solution was to throw out all of the hardware except the 286 and hook up a bunch of dumb terminals instead. The system, written in Clipper,

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

would then be tweaked to run on FoxBASE in Unix.

Hmmm. Maybe. I pointed out that the client was growing fast, and that what I thought he actually needed was a real Novell-type network running on a decent dedicated server with 286 workstations—a solution I had successfully implemented repeatedly (even though that fact doesn't necessarily make it optimal).

In spite of my serious reservations, the new vendor assured me they could hook up 32 dumb terminals, all running off that poor old 286, and that the system would run just fine. I was willing to cut the Unix folks some slack for how wonderful Unix is and how much more efficient it is, but this strained my credulity. The new hardware vendor told me he would be willing to handle the conversion to FoxBASE. In fact, he had "a team of programmers standing by" to do just that. Yeah, sure. Turned out that he knew some guys two states away that do FoxBASE on Unix.

The hardware vendor convinced the client that I was dragging my feet, and the client called me one day, after drinking heavily, and demanded that I turn over the source code. He must have expected an argument, but he was wrong.

"Excellent idea. I've always told you that you can have the source code any time you want. It's yours. I'll prepare the disks right now and send them over to you in tomorrow's mail. Of course, you realize that as soon as I turn over the source code, I will no longer be responsible for it."

He wanted to argue some more, but it's hard when the other guy won't argue back. I enclosed a letter thanking my client very much for his business over the years, expressing regret that he was having difficulty with his system, and pointing out that by his accepting the code for third-party modification I was no longer responsible for it. I was relieved to be rid of a client that had been a problem for years.

The point of this anecdote is that the issue was not as simple as buying a TV, and it was necessary to make a reasonable guess as to whether the proposed solution was workable. I didn't go along with the dumb terminal idea that the client wanted because I thought it was going to lead to more problems. My error, though, was in not saying "no" before the first hardware vendor fixed him up with the cheapo LAN. At the time, though, I didn't know that the simple LAN wouldn't work because I had never tried one. The vendor claimed that they used one themselves and that it performed flawlessly. Right.

PIRACY

One ethical issue that is difficult to deal with is piracy. There's often a conflict between legality (or at least behavior defined as legal by certain software manufacturers) and practicality. Suppose you're involved with support for a large office. You get a call that Ms. Smith needs help installing 123. When you get there, Ms. Smith has some disks that Mr. Jones made for her. Uh oh.

This one is easy. Don't do it. Insist that you have the original disks and documentation before you do an install. Make it clear to the user that they'll have to go through channels and procure a legal copy or, if your contract allows, you'll procure a legal copy for them. This won't stop Ms. Smith from borrowing Mr. Jones' entire package, but at the least it should make Mr. Jones nervous about his original disks. Decide your own limits with regard to inquiries—you're not the software police, but you should be reasonably diligent.

SHAREWARE

What about shareware? I don't have a good answer for you. It's a fact that many people use shareware without paying for it, so installing shareware for a client on a trial basis often results in financial futility for the publisher. The client will use the package, but never pay for it. Still, most companies won't buy 50 copies of something they've never heard of, even if it costs only $35—they have to try it first, so getting a package in the door may prove beneficial to the publisher in the long run.

Many government agencies are notorious abusers of shareware, while others simply prohibit it entirely. I've seen instances where no one really knows whether there's a site license for a particular package or not; one branch may have negotiated and paid for a license for perhaps 70 users in their area only. Other branches use it, blissfully unaware that the DP department didn't purchase a site license for the whole agency.

PROPRIETARY LIBRARIES

OK, what about using a proprietary library? In my opinion, you probably just stepped over the line. I strongly believe that the job of a consultant is to leave a client self-sufficient, not crippled. If you plan to use a proprietary package that the client does not have, they need to agree to this beforehand. If they don't even own a copy of Clipper, for example, they probably won't care that you used a graphic library to produce those spiffy charts while saving them time and money.

On the other hand, one of my clients and I were most annoyed to find that a previous contractor had built a system using a proprietary C library that they

A Difference That Must be Seen!

Diff*it™

The difference program that you've been waiting for, you've seen the others, now see the difference.

- △ *Unlimited file sizes.*
- △ *Scrollable split screen display.*
- △ *Display in ascii or hex.*
- △ *Configurable compare algorithm.*
- △ *25, 43 and 50 line display modes.*
- △ *Command line interface.*
- △ *Pop-up menu interface.*
- △ *Pop-up help screens.*
- △ *Optional output to disk file.*
- △ *Fast Terse mode.*
- △ *Line, Word and Byte compare.*
- △ *Batch File drive-able.*

Requires IBM PC or 100% compatable, 148KB memory, DOS 2.0 or higher, floppy or hard disk. Not copy protected. 30 day money back guarantee, send check or money order. $129.95

ORDER NOW!

DOUBLE R SOFTWARE, INC.
RD 1, Fifth Ave, Valley Forge Manor
Phoenixville, PA 19460
(215) 666-6104

Circle 31 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

happened to have at the time and had since forgotten even what it was called (the particular employee had left and taken his personal copy with him). This was not only incompetent, it was most unethical of the company.

When they did locate a copy of the library, though, they refused to deliver it because they didn't own it. This was, of course, correct; you can't hand over someone else's property even if you happen to have a stolen copy of it on hand. So, having used something that they shouldn't have (in this particular environment) to build source code that they were required by contract to turn over, and finally figuring out what it was, they then held up the project further by (correctly) not turning over the pirated copy. The client was outraged and began legal proceedings over a $200 package.

We ordered a copy of the software in the meantime with the client's permission, and billed them for it when it arrived. As a practical matter, I might have been willing to bend the law by starting work with the pirated copy before the ordered copy arrived. In that case it would have been easy for the client to cancel the order after they got their hands on the pirated copy and screw up the legal acquisition. Consequently, it was probably better the way it finally worked out.

Here's another interesting situation. Suppose you use a software package that you own to produce a product for a client. Is this wrong? Can you get into trouble over this? Well clearly some packages are supposed to be used in exactly this manner. Clipper is an example. You compile programs and turn over the EXE (and the source if your clients know what's good for them). That's why they make products like Clipper. Incidentally, this is an excellent reason for using compilers instead of interpreters like dBASE that must be purchased by the client.

THE RAMIFICATIONS

So what's the bottom line here? I don't know much about the formal study of ethics, but I do know some of the kinds of things that get consultants in trouble, and they are the kinds of things discussed above. You have to forge your own sense of ethics; perhaps you're appalled at my situational ethics and prefer a rigid, unbending set of guiding principles. Do what seems right to you, but always consider the impact that it will have on the client and on your consulting or contracting practice.

Think about the ramifications as you take that little misstep that you know is wrong, but not too wrong, and consider where your minor fooling around is likely to take you. On the other hand, if you like excitement, want to see your picture in the paper, and enjoy testifying in your own defense, then do whatever you want to. Here in Washington, D.C., on this rainy day, you've got plenty of nervous company. PJ

The Software Engineering Store

C

Borland Turbo C v2 Prof. $175
Blaise C Tools Plus/6.0 ..... $119
Clear Clear Plus for C $169
Creative Vitamin C $169
Genus PCX Prgmr Toolkit $155
manipulate ".pcx" graphics files
Gimpel PC Lint $99
Greenleaf Comm Library $179
Island Systems C Menu $129
Microsoft C v5.1 ..... $325
Microsoft Quick C $73
Novell Btrieve $199
Sequitor Codebase 4 ..... $249
read, write & create dBase III and Clipper files
Wordcraft C Workshop $69
C training manual & software

Other Tools

Digitalk Smalltalk/286 ..... $164
Burton Tlib $89
source code management system
California Scientific Brainmaker $159
neural network development system
Genus PCX Toolkit $155
Helios Proteus $169
Matrix Layout v2 ..... $175
Microsoft Macro Assembler $111
Microsoft Windows SDK v2.1 ..... $349
Mortice Kern MKS Toolkit v3 ..... $219
KORN shell & 130 UNIX™ -like utilities
Periscope I 512K v4 ..... $499
Qualitas 386-Max $69
80386 memory manager
Quarterdeck API Ref & C Library .. $179
Solution Systems Brief ..... $call
Turbo Power Turbo Professional $99
Whitewater ACTOR $449
High productivity development for MS Windows

C++ Version 2

Zortech C++ is the only true C++ compiler available for MSDOS. Now includes AT&T C++ v2 features including multiple inheritance and type safe linking. C++ debugger lets you view objects instead of translated C code.
C++ v2 ..... $175
C++ Debugger $129
C++ Developer's Edition $399

The Software Engineering Store

1-800-443-7176

In Calif: (415)957-0111

Visa • MasterCard • AMEX
COD • Check • P.O.

- Most orders shipped within 24hrs
- Free shipping on orders over $100

657 Mission Street, #300-PJ
San Francisco, CA 94105

Free Newsletter

Call or write for a catalog and your free subscription to our newsletter.

Prices subject to change without notice - ©1989 TSF All Rights Reserved

Circle 98 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

By Michael Abrash

Faster ellipses and more

Drawing still faster ellipses and more on the VGA

Do you believe in coincidence? You don't have to, you know; there's no rule that says there has to be any such thing, and many people choose to believe otherwise. They think that all events are interrelated; what appears to be coincidence actually reflects deeper meaning in the universe. Personally, I prefer to believe in random coincidence, because if coincidences are meaningful, then the universe has been trying to send me a very meaningful message lately—and while I can't imagine what that message could possibly be, I'm not completely sure I want to find out!

To wit: last time, I started by telling you that I seemed to be encountering faster circle implementations almost daily, mentioning Hal Hardenbergh's very fast, 16-bit-integer-only circle-drawing approach. A few days later, in what was either one incredible coincidence or a telegram from the Twilight Zone, a draft of an article by Tim Paterson crossed my desk. The topic of the article just happened to be drawing circles. Fast. And drawing them with no multiplies and no divides and plain old 16-bit integers.

Contributing Editor Michael Abrash works for Metagraphics of Scotts Valley, California as a senior engineer. He is the author of Power Graphics Programming *and* Zen of Assembly Language. *Readers may write to Michael at P.O. Box 414, Burlington, VT 05402-0414, or contact him via MCI Mail, user name* MABRASH.

SOUND FAMILIAR?

Tim covers ground that we've already been over, but he looks at circle drawing from a different perspective. Although the code is in C, you could convert it to assembly language easily enough. I don't have a publication date; however, Tim's article is likely to show up in *Dr. Dobb's Journal* sometime fairly soon. Keep an eye out for it.

One of the things I love about graphics is that there's always more to learn about just about any aspect of the field. (Sometimes it's also one of the things I hate about graphics;as soon as you think you're an expert, along comes someone to prove you wrong.) In case Tim and Hal haven't finally and forever nailed this one to the wall, I officially declare open season: if you know of a still faster way to draw circles, send it to me, and I'll share your idea with *PJ* readers.

Your only reward will be that you'll become locally world-famous (this is a wonderful phrase a professor of mine invented to describe people who were less famous than Bill Gates or Bill the Cat but more famous than John Q. Public) to *PJ* readers—but, hey, that's a whole lot better than a poke in the eye with a sharp stick, now, isn't it?

Let's draw some ellipses. Fast. Very fast, in fact. But, as usual, I'm sure there's something faster yet; in my experience, there always is.

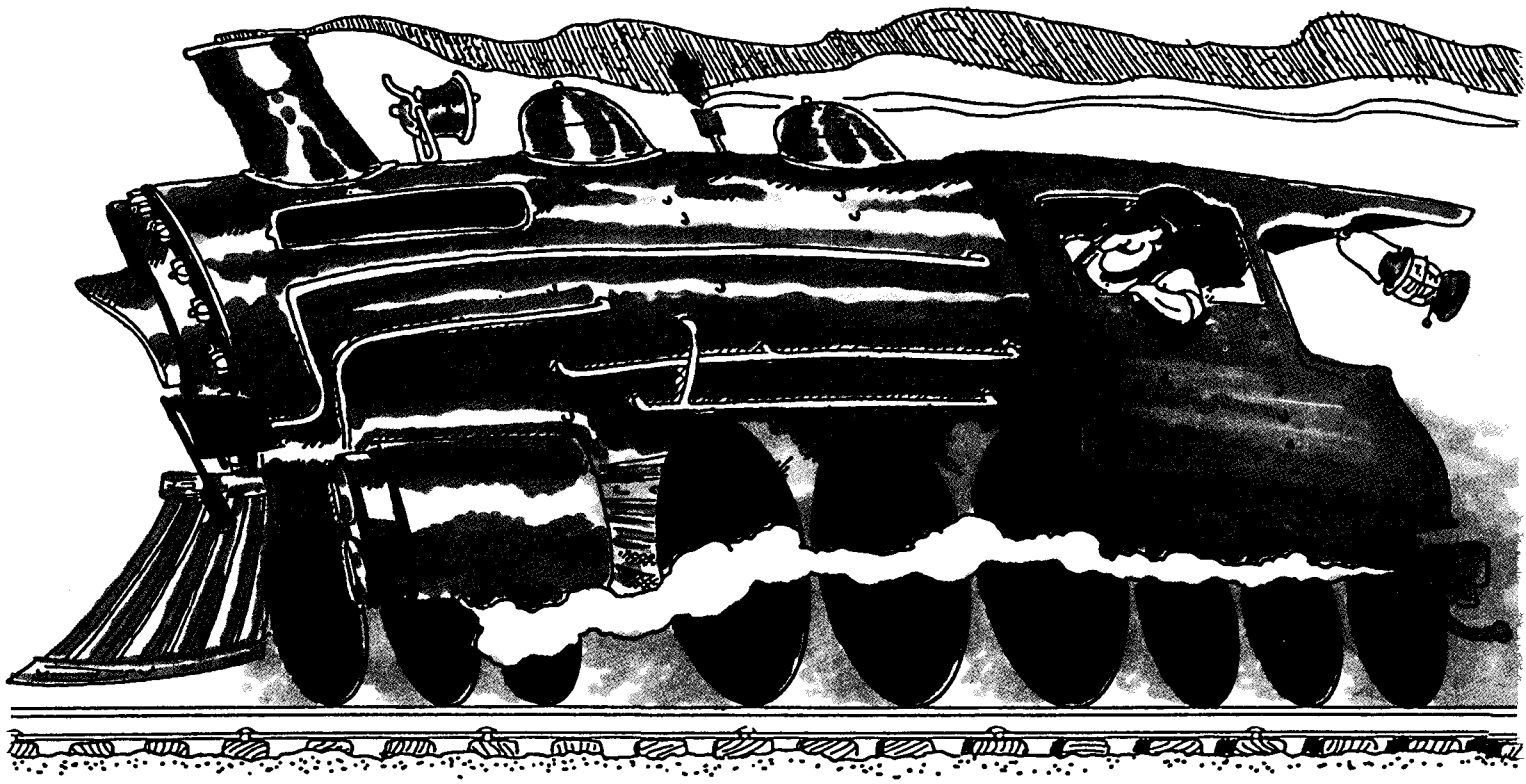
ELLIPSES, CONTINUED

Last time we learned how to draw ellipses without using any multiplies, divides, or floating-point numbers, thanks to an integer-only incremental algorithm. This time I'll better attune that code to the hardware of the EGA and VGA by eliminating the separate calculation of the screen address for each and every point. Instead, I'll generate for an octant a draw list, a set of commands to either advance or not advance along the minor axis each time drawing advances one pixel along the major axis.

Next, I'll draw the four symmetries of that draw list (the four octants which share the same major axis—that is, advance more rapidly along the same axis) one at a time, using specialized code that calculates the bit mask and offset for each pixel as a function of the last pixel, thereby eliminating all multiplications and multi-bit rotations. I'll then repeat the process for the four octants in which the other coordinate is the major axis.

I'll also rewrite in assembly language the code that generates draw lists. The assembly language code that draws octants from draw lists was in installment #17 (*PJ* 8.2, March 1990), and it's

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



several hundred lines long—long enough that it was almost cut from the magazine in installment #17. I'm certainly not going to push my luck by asking to have it reprinted. However, the full code is available on the listing disk and on *PJ*'s bulletin board.

I do apologize for being unable to present the full code here and for referring back to other installments so frequently rather than explaining principles and code again—in other words, for failing to make this installment self-contained, which can be a nuisance for newcomers to this column. However, back issues are available, and by being able to refer back to and build on previous installments, I'm able to tackle more complex and rewarding topics than is possible in a single article.

I assure you that any stand-alone article about drawing ellipses would either skip a great many details or come nowhere near the performance of the final ellipse-drawing code in this installment; in contrast, if you've followed this column from installment #16 forward, you know the principles of circle and ellipse drawing **and** you have high-performance code to go with that knowledge.

If you've followed the column from installment #1, you also understand issues such as set/reset, the bit mask, and write mode 3 that play an important part in the listings in this installment. EGA/VGA programming requires a broad knowledge base, one that "On Graphics" has spent years building.

ELLIPSE DRAWING MADE FAST

Let's get on to the code. Listing 1 is C ellipse-drawing code, differing from Listing 4 in installment #18 (henceforth called Listing 18-4) in that Listing 1 generates a draw list and draws the four symmetries of that list, then does it all again for the other axis, as described above. That change produces a performance improvement of about 35%, as shown in Table 1. Listing 2, a sample C program that can call any of our ellipse-drawing functions, is used to time the various implementations. Because this installment brings together a lot of familiar stuff, I made the sample program a little more interesting than the standard concentric ellipses. A circle with pointy ears, it gives you an idea of the sorts of shapes easily created with sets of ellipses; run it yourself and see.

By the way, I am well aware that Listing 1 is not fully optimized; for one thing, it doesn't take advantage of write mode 3 and pixel-accumulation techniques used by the assembly code in Listing 4 (in a portion of the listing that's not printed in this issue, alas). Listing 1 is intended as an illustrative bridge between the standard ellipse-drawing code of the last installment and the fast but hard-to-understand code in Listing 4, so I leaned toward comprehensibility rather than maximum speed in Listing 1. To my mind, it's wasted effort to spend time squeezing cycles out of Listing 1 when Listings 3 and 4 will always be faster, no matter how well we optimize Listing 1.

Listing 3 is the C portion of the final and fastest ellipse-drawing routine. Listing 3 calls functions in the assembly-language module shown in Listing 4 in order to perform the two critical aspects of ellipse-drawing—draw list generation and the actual octant drawing. (As explained above, Listing 4 shows only the code that generates the draw list for an ellipse octant; the code that actually draws an octant from a draw list appeared in installment #17 and can be obtained from *PJ* in a variety of media.)

Listings 3 and 4 together are about twice as fast as Listing 1, and about 2.5 to 3.5 times faster than Listing 18-4. Lastly, they are 60 or so times faster than the floating-point based implementation in Listing 18-1.

See what a little forethought and attention to detail can do?

OPTIMIZING THE CODE

This installment is truly a synthesis of what has come before. We spent the last installment developing the incremental ellipse-drawing approach, and the installment before that developing the draw-list approach. The only new ele-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ment in this installment is the assembly language code to generate a draw list, and that's a straightforward port of the C code in Listing 1, for the most part.

I must in all honesty point out that it's scarcely worth bothering with converting the draw list generation code to assembly language at all. First, drawing from the draw list is likely to take much longer than generating the draw list, because four octants are drawn for each draw list generated and drawing is usually slowed considerably by video wait states. Draw list generation doesn't represent a very large fraction of total execution time, and therefore isn't a particularly fruitful place to expend optimization effort.

Second, draw list generation involves many 32-bit variables, too many to be able to keep them all in the registers; when that happens, particularly in fairly straightforward add-subtract-compare code like draw list generation, in my experience there's not likely to be a sizeable difference between good assembly code and the code produced by a good C compiler. Sure, the assembly-language code is better, but once again the difference isn't likely to translate into a sizable improvement in overall performance.

When you optimize code, it's important to understand where in your code the effort will pay off best. In Listings 3 and 4, optimization effort is better spent in trying to draw octants from draw lists faster than in trying to generate draw lists faster.

PERISCOPE IV

THE REAL-TIME SOURCE-LEVEL DEBUGGER

With Periscope Model IV, you can debug your software while it runs at full speed, something *no* software-only debugger can do.

You can also examine the execution history of a hardware interrupt in Periscope IV's real-time trace buffer, something *else* no software-only debugger can do.

When The Going Gets Tough... Periscope IV helps you debug most any program, but is specially designed to debug programs that software-only debuggers can't. Model IV users use it to debug TSRs, ISRs, device-drivers, DOS, BIOS, communications software, real-time data acquisition programs, multitasking software, network software, keyboard drivers, disk caching software, systems software, spread sheets, EMM products, and so forth.

"Periscope is my #1 favorite program... I would rather change my editor than my debugger," writes Phil Mayes, who used Periscope IV to track down some very elusive bugs causing intermittent corruption.

Periscope IV provides source-level support for popular PC compilers and linkers, such as those produced by Microsoft, Borland, and others, and it runs on most any 80286 or 80386 at speeds up to 25MHz.

The new Remote Debugging feature lets you use Periscope IV to debug programs running on IBM PS/2s and compatibles. The optional Plus board keeps Periscope from using any of the lower 640K, so Periscope can't be overwritten and doesn't use the memory your program needs.

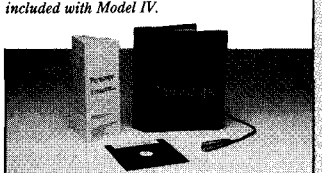
You can try Periscope IV for ten days at no charge before you buy it. Call our toll-free number for details.

FREE 10-day Evaluation

Periscope IV prices range from $2195 for a 16 MHz 80286 system to $2995 for a 25 MHz 80386 system. The optional 512K Periscope PLUS board is $400.

Given the value of your time, can you afford *not* to try it?

The Periscope manual, software, breakout switch and quick-reference card are included with Model IV.



"The Model IV hardware really makes Periscope shine, especially when you've got timing-related problems. I can now track down changing pointers and altered buffers on my 386. I've been using it to debug Crosstalk® Mk. 4 and there's just no better way to do it."

JEFF GARBERS
Director of Software Development
Crosstalk Communications

Call Toll-Free Today for More Information or to Order Your Periscope: 800-722-7006

In the UK call Roundhill Computer Systems at 0672 84 535. In Germany call H+B EDV at 07542 6353 or ComFood at 02534 7093. In Sweden call LinSoft at 013 124780.

The Periscope Company, Inc.
1197 Peachtree St., Plaza Level
Atlanta, GA 30361
404 875-8080 • FAX 404 872-1973

AIMING DOWN THE MIDDLE

There is one slightly tricky element to Listing 4. If you look closely, you'll note that each draw list element is **always** set to 0. Later, that same element may be set again, this time to 1, meaning that a single element may be set twice, incurring an extra instruction and an extra memory access. Is this wise? In this case, it probably is, although the truth of the matter is far from clear; the question illustrates the hazards of optimizing in today's multi-platform (8088, 286, 386, and 486 with a variety of memory architectures) world.

Presetting each element to 0 allows us to branch and be done with it if there is no minor move, although it also requires us to reset each element for which there **is** a minor move. The alternative would be something like:

```
    js   NoMoveMinor
<advance minor coordinate>
    mov [di],1
    jmp short  MoveMajor
NoMoveMinor:
    mov byte ptr [di],0
MoveMajor:
    inc  di
```

This code only sets each draw list element once, but also requires a branch in the case where the minor coordinate advances. So the latter approach saves one **mov [di],0** when the minor coordinate advances, but adds one **jmp short MoveMajor** at the same time. That's not a good trade on any 80x86 processor; it's a terrible one on the 286 and 386, where **mov [di],0** takes 2 cycles and **jmp short MoveMajor** takes 9 cycles or so. Although in actual use instruction-fetching can alter those cycle counts somewhat, any alteration is

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

usually to the relative detriment of **jmp**, which empties the prefetch queue.

Well, then, why not load 1 or 0 into a register and then store the register, eliminating a memory access? That code would look like this:

```
    sub  al,al;assume there's no minor move
    js   NoMoveMinor
<advance minor coordinate>
    inc  ax;upper byte doesn't matter, but word INC
         ; is more efficient than byte INC
NoMoveMinor:
    stosb
```

This last approach might be faster—or it might not. It all depends on the processor and the memory architecture. On an 8088, there's no question but that the last approach is faster; the 8088 instruction set favors both string instructions like **stosb** and keeping values in registers. On a 386 machine, however, **mov [di],0** takes just two cycles, and **inc di** takes another two; **stosb** takes four, so there's no advantage to **stosb** there. However, the last approach requires not only **stosb** but also **sub al,al** and possibly **inc ax**, so it's actually two cycles slower on a 386 in both cases.

TABLE 1: EXECUTION TIMES (SECONDS)

	Processor	
Listing	**286**	**386**
18-1	1088	406
(C/float no 80X87)		
18-4	46	17
(C/integer)		
19-1	33	13
(C/incremental pixel)		
19-3/4	14	7
(ASM,ISVGA=0)		
19-3/4	13	5
(ASM/write mode 3,ISVGA=1)		

Execution times when linked to Listing 2. Listings **18-n** are from installment #18. C code was compiled with Turbo C with optimization (-G -O -Z -r). The 286 machine was a 10 MHz clone (1 wait-state) with a 16-bit Video Seven VRAM VGA. The 386 was a 20 MHz Toshiba 5200 (32K static RAM cache) with a built-in Paradise VGA (mono adapter also installed).

This gets us into issues such as whether the system has a cache, and if so whether the code is in the cache and whether it's a write-back or write-through cache, and if not whether we've hit a ready-to-go interleaved memory bank or the current column in static-column RAM. There's no clear answer as to which code is fastest unless you know not only your target processor but also your target memory architecture.

The only thing that's constant across all 80x86-family processors is that branching is slow, so all of the alternative approaches we've discussed are faster than the obvious approach that we first discussed: the approach of not preloading or presetting at all and branching in both cases.

I'd like to be able to pull a clear, simple lesson out of all this, but the life of an optimizing PC programmer is neither simple nor clear. If there's a

ProGraphx

ProGraphx Menus™

Simply add four lines of code to your program and you can have a pull-down or pull-up menu system. Define a menubar with a text file, compile it and receive events calls back to your program. Change the menubar without recompiling your program to create demo versions or other adaptations. A sample application of a Microcoded processor is included along with its two editors.

ProGraphx Menus™	$99
Menus with Source	199

ProGraphx Toolbox™

Over 50 hi-speed graphics routines for EGA/VGA 16 color modes, including proprietary icon routines to display color images.

ProGraphx Image Editor, a complete color icon paint package to edit and view up to 50 icons at once. Simply paint an icon, then use the ***ProGraphx Toolbox™*** routines to display it with amazing speed. Over **500** sample icons for use in your own programs. ***Operation:Airlift***, a full screen scrolling arcade style game with source code.

All routines written in assembly for fastest possible performance.

ProGraphx ToolBox™	$199
Toolbox with Source	399

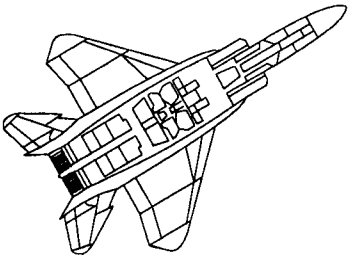
Stanwood Associates, Inc.
303 East Ohio Street Chicago, IL 60611

1-(800) 752-7982

(312) 828-9734 VISA/MC FAX (312) 828-9742

ProGraphx
Speed, Simplicity, and Power for Graphics Programmers

Compatible with Turbo C or Pascal, Zortech C/C++, Microsoft C or any language compatible with 80x86 object code. Uses less than 10K of your code segment!



Also available from :
Programmer's Shop &
Programmer's Connection

Circle **330** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

NEW

Develop Powerful Image-based Applications Quickly and Easily

Introducing VICTOR, video capture and image processing library

VICTOR simplifies development of scientific imaging, quality control, security, and image database software. Victor gives you complete control of your video frame grabber and also includes image processing, display, and TIFF/PCX file handling routines.

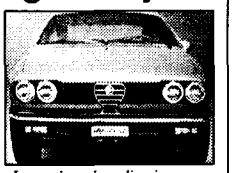
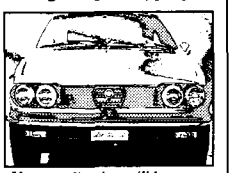


Image-based applications can be developed in QuickC and Turbo C environments.

Application development is simplified because we've taken care of the details of device control and image processing. All the hard low level coding has been done -- you don't have to worry about bit twiddling, video timing, etc. This means you can concentrate on your application. Your software can have features such as: live video on VGA, zoom, display multiple images with text and graphics, pixel editing and image processing. And, to get you up and running fast, we've included Zip Image Processing software for rapid testing and prototyping of image processing and display functions.

Victor supports Microsoft C, QuickC, and TurboC, includes over 100 functions, demonstration and prototyping software, full documentation, and source code for device control routines . . . all for only $195. Full source code add $100, video frame grabbers available.



Your application will have powerful image handling capabilities.

Call (314) 962-7833 to order
VISA/MC/COD

Catenary Systems 470 Belleview St Louis MO 63119 (314) 9627833

Circle **321** on Reader Service Card

PRINTER GRAPHICS IN HIGH RESOLUTION!

GRAPHLINK™ Printer Graphics Toolkit

GraphLink™ lets your programs tap the *full potential* of printers to produce dazzling *printed* graphics in *high-resolution!*

- Uses the PRINTER'S resolution, not the SCREEN'S!
- Supports the most popular laser, inkjet, and dot matrix printers
- 80+ routines in a linkable library
- Fully scalable fonts
- Emulates the Borland Graphics Interface - great for BGI programs!
- Works with all screen graphics libraries
- Can *automatically* build the printer image
- Supports Turbo Pascal 4.0 and higher
- Available soon for Microsoft C, Quick C, and Turbo C
- *NO ROYALTIES!*
- Only ***$199*** (shipping: USA $5, Canada $10, Other $15) (with Source Code only $299)

30 Day No Risk Money-Back Guarantee

GRAPHLINK
The Printer Graphics Solution™

VISITECH SOFTWARE
D5 3807 Ridgewood Court
Pittsburgh, PA 15239 USA
(412) 733-4775

LISTING 1

```
/*
  *
  * Draws an ellipse of the specified X and Y axis radii and color,
  * using a fast integer-only & square-root-free approach, and
  * generating the arc for one octant into a buffer, drawing four
  * symmetries from that buffer, then doing the same for the other
  * axis.
  * Compiles with either Turbo C 2.0 or MSC 5.0.
  * VGA or EGA.
  */
#include <dos.h>

/* Handle differences between Turbo C & MSC. Note that Turbo C accepts
   outp as a synonym for outportb, but not outpw for outport */
#ifdef __TURBOC__
#define outpw  outport
#endif

#define SCREEN_WIDTH_IN_BYTES 80      /* # of bytes across one scan
                                         line in mode 12h */
#define SCREEN_SEGMENT        0xA000 /* mode 12h display memory seg */
#define GC_INDEX              0x3CE  /* Graphics Controller port */
#define SET_RESET_INDEX       0      /* Set/Reset reg index in GC */
#define SET_RESET_ENABLE_INDEX 1     /* Set/Reset Enable reg index
                                         in GC */
#define BIT_MASK_INDEX        8      /* Bit Mask reg index in GC */

unsigned char PixList[SCREEN_WIDTH_IN_BYTES*8/2];
                                     /* maximum major axis length is
                                        1/2 screen width, because we're
                                        assuming no clipping is needed */

/* Draws the arc for an octant in which Y is the major axis. (X,Y) is the
   starting point of the arc. HorizontalMoveDirection selects whether the
   arc advances to the left or right horizontally (0=left, 1=right).
   RowOffset contains the offset in bytes from one scan line to the next,
   controlling whether the arc is drawn up or down. Length is the
   vertical length in pixels of the arc, and DrawList is a list
   containing 0 for each point if the next point is vertically aligned,
   and 1 if the next point is 1 pixel diagonally to the left or right */
void DrawVOctant(int X, int Y, int Length, int RowOffset,
     int HorizontalMoveDirection, unsigned char *DrawList)
{
    unsigned char far *ScreenPtr, BitMask;

    /* Point to the byte the initial pixel is in */
#ifdef __TURBOC__
    ScreenPtr = MK_FP(SCREEN_SEGMENT,
                (Y * SCREEN_WIDTH_IN_BYTES) + (X / 8));
#else
    FP_SEG(ScreenPtr) = SCREEN_SEGMENT;
    FP_OFF(ScreenPtr) =(Y * SCREEN_WIDTH_IN_BYTES) + (X / 8);
#endif
    /* Set the initial bit mask */
    BitMask = 0x80 >> (X & 0x07);

    /* Draw all points in DrawList */
    while ( Length-- ) {
       /* Set the bit mask for the pixel */
       outp(GC_INDEX + 1, BitMask);
       /* Draw the pixel. ORed to force read/write to load latches.
          Data written doesn't matter, because set/reset is enabled
          for all planes. Note: don't OR with 0; MSC optimizes that
          statement to no operation */
       *ScreenPtr |= 0xFF;
       /* Now advance to the next pixel based on DrawList */
       if ( *DrawList++ ) {
          /* Advance horizontally to produce a diagonal move. Rotate
             the bit mask, advancing one byte horizontally if the bit
             mask wraps */
          if ( HorizontalMoveDirection == 1 ) {
             /* Move right */
             if ( (BitMask >>= 1) == 0 ) {
                BitMask = 0x80;   /* wrap the mask */
                ScreenPtr++;      /* advance 1 byte to the right */
             }
          } else {
             /* Move left */
             if ( (BitMask <<= 1) == 0 ) {
                BitMask = 0x01;   /* wrap the mask */
                ScreenPtr--;      /* advance 1 byte to the left */
             }
          }
       }
```


Circle **304** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        ScreenPtr += RowOffset; /* advance to the next scan line */
    }
}
/* Draws the arc for an octant in which X is the major axis. (X,Y) is the
    starting point of the arc. HorizontalMoveDirection selects whether the
    arc advances to the left or right horizontally (0=left, 1=right).
    RowOffset contains the offset in bytes from one scan line to the next,
    controlling whether the arc is drawn up or down. Length is the
    horizontal length in pixels of the arc, and DrawList is a list
    containing 0 for each point if the next point is horizontally aligned,
    and 1 if the next point is 1 pixel above or below diagonally */
void DrawHOctant(int X, int Y, int Length, int RowOffset,
    int HorizontalMoveDirection, unsigned char *DrawList)
{
    unsigned char far *ScreenPtr, BitMask;
    /* Point to the byte the initial pixel is in */
#ifdef __TURBOC__
    ScreenPtr = MK_FP(SCREEN_SEGMENT,
                (Y * SCREEN_WIDTH_IN_BYTES) + (X / 8));
#else
    FP_SEG(ScreenPtr) = SCREEN_SEGMENT;
    FP_OFF(ScreenPtr) =(Y * SCREEN_WIDTH_IN_BYTES) + (X / 8);
#endif
    /* Set the initial bit mask */
    BitMask = 0x80 >> (X & 0x07);
    /* Draw all points in DrawList */
    while ( Length-- ) {
       /* Set the bit mask for the pixel */
       outp(GC_INDEX + 1, BitMask);
       /* Draw the pixel (see comments above for details) */
       *ScreenPtr |= 0xFF;
       /* Now advance to the next pixel based on DrawList */
       if ( *DrawList++ ) {
          /* Advance vertically to produce a diagonal move */
          ScreenPtr += RowOffset; /* advance to the next scan line */
       }
       /* Advance horizontally. Rotate the bit mask, advancing one
          byte horizontally if the bit mask wraps */
       if ( HorizontalMoveDirection == 1 ) {
          /* Move right */
          if ( (BitMask >>= 1) == 0 ) {
             BitMask = 0x80;   /* wrap the mask */
             ScreenPtr++;      /* advance 1 byte to the right */
          }
       } else {
          /* Move left */
          if ( (BitMask <<= 1) == 0 ) {
             BitMask = 0x01;   /* wrap the mask */
             ScreenPtr--;      /* advance 1 byte to the left */
          }
       }
    }
}
/* Draws an ellipse of X axis radius A and Y axis radius B in
  * color Color centered at screen coordinate (X,Y). Radii must
  * both be non-zero */
void DrawEllipse(int X, int Y, int A, int B, int Color) {
    int WorkingX, WorkingY;
    long Threshold;
    long ASquared = (long) A * A;
    long BSquared = (long) B * B;
    long XAdjust, YAdjust;
    unsigned char *PixListPtr;

    /* Set drawing color via set/reset */
    outpw(GC_INDEX, (0x0F << 8) | SET_RESET_ENABLE_INDEX);
                                /* enable set/reset for all planes */
    outpw(GC_INDEX, (Color << 8) | SET_RESET_INDEX);
                                /* set set/reset (drawing) color */
    outp(GC_INDEX, BIT_MASK_INDEX); /* leave the GC Index reg pointing
                                       to the Bit Mask reg */
    /* Draw the four symmetric arcs for which X advances faster (that is,
       for which X is the major axis) */
    /* Draw the four arcs; set draw parameters for initial point (0,B) */
    /* Calculate all points along an arc of 1/8th of the ellipse and
       store that info in PixList for later drawing */
    PixListPtr = PixList;
    WorkingX = 0;
    XAdjust = 0;
    YAdjust = ASquared * 2 * B;
    Threshold = ASquared / 4 - ASquared * B;

    for (;;) {
       /* Advance the threshold to the value for the next X point
          to be drawn */
       Threshold += XAdjust + BSquared;
       /* If the threshold has passed 0, then the Y coordinate has
          advanced more than halfway to the next pixel and it's time
          to advance the Y coordinate by 1 and set the next threshold
          accordingly */
       if ( Threshold >= 0 ) {
          YAdjust -= ASquared * 2;
          Threshold -= YAdjust;
          *PixListPtr++ = 1;   /* advance along both axes */
       } else {
          *PixListPtr++ = 0;   /* advance only along the X axis */
       }
       /* Advance the X coordinate by 1 */
       XAdjust += BSquared * 2;
       WorkingX++;
       /* Stop if X is no longer the major axis (the arc has passed the
          45-degree point) */
       if ( XAdjust >= YAdjust )
          break;
    }
    /* Now draw each of 4 symmetries of the octant in turn, the
       octants for which X is the major axis. Adjust every other arc
       so that there's no overlap */
    DrawHOctant(X,Y-B,WorkingX,SCREEN_WIDTH_IN_BYTES,0,PixList);
    DrawHOctant(X+1,Y-B+(*PixList),WorkingX-1,SCREEN_WIDTH_IN_BYTES,1,
       PixList+1);
    DrawHOctant(X,Y+B,WorkingX,-SCREEN_WIDTH_IN_BYTES,0,PixList);
    DrawHOctant(X+1,Y+B-(*PixList),WorkingX-1,-SCREEN_WIDTH_IN_BYTES,1,
       PixList+1);
    /* Draw the four symmetric arcs for which X advances faster (that is,
       for which Y is the major axis) */
    /* Draw the four arcs; set draw parameters for initial point (A,0) */
    /* Calculate all points along an arc of 1/8th of the ellipse and
       store that info in PixList for later drawing */
    PixListPtr = PixList;
    WorkingY = 0;
    XAdjust = BSquared * 2 * A;
    YAdjust = 0;
    Threshold = BSquared / 4 - BSquared * A;

    for (;;) {
       /* Advance the threshold to the value for the next Y point
          to be drawn */
       Threshold += YAdjust + ASquared;

       /* If the threshold has passed 0, then the X coordinate has
          advanced more than halfway to the next pixel and it's time
          to advance the X coordinate by 1 and set the next threhold
          accordingly */
       if ( Threshold >= 0 ) {
          XAdjust -= BSquared * 2;
          Threshold = Threshold - XAdjust;
          *PixListPtr++ = 1;   /* advance along both axes */
       } else {
          *PixListPtr++ = 0;   /* advance only along the X axis */
       }

       /* Advance the Y coordinate by 1 */
       YAdjust += ASquared * 2;
       WorkingY++;

       /* Stop if Y is no longer the major axis (the arc has passed the
          45-degree point) */
       if ( YAdjust > XAdjust )
          break;
    }

    /* Now draw each of 4 symmetries of the octant in turn, the
       octants for which Y is the major axis. Adjust every other arc
       so that there's no overlap */
    DrawVOctant(X-A,Y,WorkingY,-SCREEN_WIDTH_IN_BYTES,1,PixList);
    DrawVOctant(X-A+(*PixList),Y+1,WorkingY-1,SCREEN_WIDTH_IN_BYTES,1,
       PixList+1);
    DrawVOctant(X+A,Y,WorkingY,-SCREEN_WIDTH_IN_BYTES,0,PixList);
    DrawVOctant(X+A-(*PixList),Y+1,WorkingY-1,SCREEN_WIDTH_IN_BYTES,0,
       PixList+1);
    /* Reset the Bit Mask register to normal */
    outp(GC_INDEX + 1, 0xFF);
    /* Turn off set/reset enable */
    outpw(GC_INDEX, (0x00 << 8) | SET_RESET_ENABLE_INDEX);
}
```

LISTING 2

```
/*
  * Draws ellipses of varying eccentricities in the VGA's hi-res mode,
  * mode 12h.
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MORE 34010 Video Co-Processors from PC Tech

- Mono 736 x 1024 - 2048 x 1536
- Color 1024 x 768 (1280 x 1024 soon!)
- 512K - 4M DRAM
- Optional CCITT & DGIS support
- **AGE XoftWare X Window Server certified board**
- C Compiler, Assembler, Linker, Debugger, DGIS Development Kit & TIGA Toolkit available
- **TIGA** included with all boards
- Custom designs available
- **PC Tech understands the 34010**

For further information, please contact:

PC Tech
907 N. 6th St.
Lake City, MN 55041
(612) 345-4555 (Voice)
(612) 345-5514 (FAX)

PC Tech is a trademark of PC Tech, Inc.

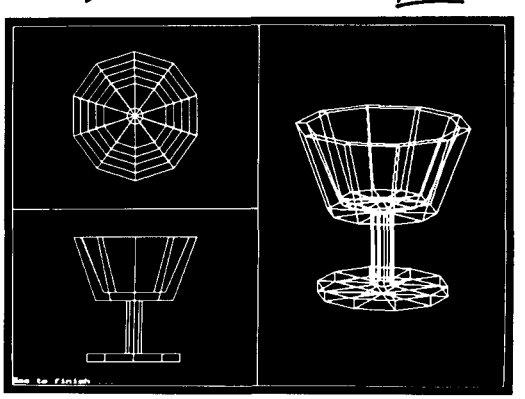
PC, XT, AT, DGIS, XoftWare, and DGIS are trademarks or registered trademarks of their respective companies.

Circle **208** on Reader Service Card

THE ULTIMATE CAD/CAM ENGINE

TurboGeometry Library 3.0!

Helps You build it right THE FIRST TIME!



Over 300 2D & 3D device independent geometric routines. Surfacing; Spherical & Solid Geometry; Tangents; Curves; Hidden Line; Polygons; Perspectives; 2D & 3D Clipping; Link List Library & more. In C or Pascal, w/source. 400 pg manual. $199.95, Foreign $225.00. Prices incl S&H. 30 day guarantee. VISA, MC, PO, Chk.

2116 E. Arapaho Rd.
Suite 487
Richardson, TX 75081
USA

Disk Software, Inc.

FAX or Phone
214-423-7288
Toll Free
1-800-635-7760

Circle **229** on Reader Service Card


```
  * For VGA only.
  * Compile and link with listingX.c (where X is 1 or 4) with:
  *    tcc listingX listing2      (Turbo C)
  *    cl listingX.c listing2.c   (MSC)
  */
#include <dos.h>
#include <stdio.h>

main() {
    int XRadius, YRadius, Temp, Color, i;
    union REGS Regs;

    /* Select VGA's hi-res 640x480 graphics mode, mode 12h */
    Regs.x.ax = 0x0012;
    int86(0x10, &Regs, &Regs);
    /* Repeat 10 times */
    for ( i = 0; i < 10; i++ ) {
       /* Draw nested ellipses */
       for ( XRadius = 319, YRadius = 1, Color = 7; YRadius < 240;
             XRadius -= 1, YRadius += 2 ) {
          DrawEllipse(640/2, 480/2, XRadius, YRadius, Color);
          Color = (Color + 1) & 0x0F;   /* cycle through 16 colors */
       }
    }
    /* Wait for a key, restore text mode, and done */
    scanf("%c", &Temp);
    Regs.x.ax = 0x0003;
    int86(0x10, &Regs, &Regs);
}
```

LISTING 3

```
/*
  * Draws an ellipse of the specified X and Y axis radii and color,
  * using a fast integer-only & square-root-free approach, and
  * generating the arc for one octant into a buffer, drawing four
  * symmetries from that buffer, then doing the same for the other
  * axis. Uses assembly language for inner loops of octant generation
  * & drawing. Link to L4.ASM, modified as described in the header of
  * that file, or link to L4FULL.ASM, available from Programmer's
  * Journal. Compiles with either Turbo C 2.0 or MSC 5.0.
  * VGA or EGA. */
#define ISVGA  0              /* set to 1 to use VGA write mode 3*/
#include <dos.h>              /* keep ISVGA synchronized with Listing 4 */

/* Handle differences between Turbo C & MSC. Note that Turbo C accepts
    outp as a synonym for outportb, but not outpw for outport */
#ifdef __TURBOC__
#define outpw  outport
#endif
#define SCREEN_WIDTH_IN_BYTES 80    /* # of bytes across one scan
                                       line in mode 12h */
#define SCREEN_SEGMENT        0xA000 /* mode 12h display memory seg */
#define GC_INDEX              0x3CE /* Graphics Controller port */
#define SET_RESET_INDEX       0     /* Set/Reset reg index in GC */
#define SET_RESET_ENABLE_INDEX 1    /* Set/Reset Enable reg index in GC */
#define GC_MODE_INDEX         5     /* Graphics Mode reg index in GC */
#define COLOR_DONT_CARE       7     /* Color Don't Care reg index in GC */
#define BIT_MASK_INDEX        8     /* Bit Mask reg index in GC */

unsigned char PixList[SCREEN_WIDTH_IN_BYTES*8/2];
/* maximum major axis length is 1/2 screen width, because we're assuming
 * no clipping is needed
 * Draws an ellipse of X axis radius A and Y axis radius B in color Color
 * color Color centered at screen coordinate (X,Y). Radii must
 * both be non-zero */
void DrawEllipse(int X, int Y, int A, int B, int Color) {
    int Length;
    long Threshold;
    long ASquared = (long) A * A;
    long BSquared = (long) B * B;
    long XAdjust, YAdjust;
    unsigned char *PixListPtr, OriginalGCMode;

    /* Set drawing color via set/reset */
    outpw(GC_INDEX, (0x0F << 8) | SET_RESET_ENABLE_INDEX);
                                /* enable set/reset for all planes */
    outpw(GC_INDEX, (Color << 8) | SET_RESET_INDEX);
                                /* set set/reset (drawing) color */
#if ISVGA
    /* Remember original read/write mode & select
       read mode 1/write mode 3, with Color Don't Care
       set to ignore all planes and therefore always return 0xFF */
    outp(GC_INDEX, GC_MODE_INDEX);
    OriginalGCMode = inp(GC_INDEX + 1);
    outp(GC_INDEX+1, OriginalGCMode | 0x0B);
    outpw(GC_INDEX, (0x00 << 8) | COLOR_DONT_CARE);
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

moral here, it would be: aim down the middle. That is, try to write code that provides good performance (by assembly language standards) on every 80x86 processor, and terrible performance on none. In this particular example, that translates into not branching any more than you absolutely must.

CATCHING UP

While we've been busy drawing vaguely roundish objects for the last few installments, things have been piling up. We're heading for an in-depth exploration of the VGA's color capabilities, but before we start that, I'm going to clear a little of the accumulated rubble away.

The first order of business is *Graphics Programming for the 8514/A*, by Jake Richter and Bud Smith (M&T Publishing, 1990). If this book were pure dreck—if it contained nothing but a summary of the 8514/A's registers and some working code—it would still be worth $29.95 because, as I write this, it is absolutely the only generally available reference about programming the 8514/A's registers. (Some people can get clone manufacturers' documentation, but they're the fortunate few.)

So the book would be worth the money if it were lousy—but as it happens it's not. It covers a lot of ground, from hardware through the registers and on up to the Adapter Interface (AI), the driver IBM wants you to program the 8514/A through. IBM didn't publish any register specifications for this board, precisely in order to force programmers to go through the AI. The idea was that AI-based graphics programs would be portable to the successor to the 8514/A.

In the PC arena, portability is generally a nice but impractical idea, and the AI is no exception. The performance of graphics programs which go through the AI is about half that of programs which use the registers directly, too much for anyone to ignore; for instance, Microsoft's Windows and Presentation Manager drivers program the registers directly. (In fact, Microsoft was the one that broke the 8514/A register-programming dam by producing a Windows driver when no one else could get access to the 8514/A register specification.)

The next time someone says assembly language graphics code is **only** twice as fast as C—implying that the difference isn't worth the trouble—remind them that many people, including Microsoft, went to great efforts to rewrite their code for the two-times advantage of programming the 8514/A's registers directly. Clearly, **speed pays**.

If you're at all concerned about performance, you'll want to program the registers directly—and to do that, right now you need either access to a clone manufacturer or to this book. In fact, unless you're interested in building an 8514/A board yourself (the primary focus of the clone manufacturers), Jake and Bud's book is the place to start.

It's superfluous to say this book is recommended. Either you need it or you don't. If you need it, run, don't walk, to get it. Enough said.

FOLLOW-UP ON EGA COLORS

Digging deeper into the pile on my desk, I found an embarrassingly old letter from Eric Klien, who asks if I could list the 64 colors the EGA is capable of displaying, as we discussed back in installment #9. (He can't see them for himself because his adapter, for some reason, is only capable of displaying 16 colors, but he wants to know what he's missing. Bear in mind that the Enhanced Color Display shows 64 colors only in 350-scan-line modes; in 200-scan-line modes, two of the color signals are ignored, and another switches to signifying intensity. Old 200-scan-line-only CGA-compatible displays can only show 16 colors, period.)

Gee, Eric, I don't even know if the colors all have names; they're just mixes of six basic color components. EGA colors are based on the red, green, blue, secondary red, secondary green, and secondary blue signals coming from one of the sixteen locations in the EGA's palette RAM look-up table, as selected for each pixel by four bits of data coming from display memory. (In text mode it's a little more complicated, but this is enough to give you the idea.)

The organization of the color information in each palette RAM location is shown in Figure 1 (reprinted from *PJ* 6.6). The combined on/off (1/0) statuses of those six components in the selected

HI-SCREEN XL™

THE PROVEN USER-INTERFACE SOLUTION FOR YOUR PROGRAMS

Since 1983, Softway has specialized in user-interface development tools aimed at demanding professional programmers.

WINDOWS • MENUS
DATA ENTRY • HELP

HI-SCREEN XL offers a high-level modular approach to screen management that provides for superior programming **control**, **flexibility** and **speed.**

- The built-in **screen editor** is used to create screens/windows/menus along with all field descriptions. Any **Hercules**, **CGA**, **EGA** or **VGA** graphics can also be captured. Screens are then saved in a compressed file format outside your program.
- **High-level calls** from your program give you total control over the user interface. Combine text and graphics, dialog boxes, context-sensitive help, elaborate data entry & menu management, keyboard & mouse control, etc.

WITH MOUSE AND GRAPHICS SUPPORT

- Supports all versions of **C**, **Basic**, **Cobol**, **Fortran**, **Pascal**, **dBase**, **Assembly**, etc.
- Use linkable (with C and QuickBasic) or removable TSR screen driver (30-40K).
- 2 compatible versions:

HI-SCREEN XL.....**$149**
HI-SCREEN XL *Professional Series*
(with mouse & graphics support):.....**$325**

30 day money-back guarantee • No royalties

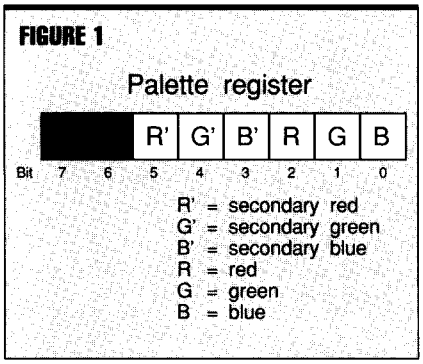
Call today: **1-800-338-2852**
in CA: 415-397-4666 • Fax: 415-397-4677

SOFTWAY, Inc.
185 Berry Street, Suite 5411, San Francisco, CA 94107

Circle **110** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 1



palette RAM location will determine the color for each pixel, with the secondary signals having less effect than the primary signals.

For example, if a given pixel's color comes from a palette RAM location containing 09h (blue and secondary blue on, all other colors off), you get bright blue. If it comes from a location containing 01h (blue only on), you get blue. If it comes from a location containing 08h (secondary blue only on), you get dim blue. If it comes from a location containing 03h (green and blue on), you get cyan. And so on.

The information above may not make perfect sense to you, because it's a continuation of the discussion of EGA colors we had back in installment #9. If you'd like more information, you'll find it in *PJ* 6.6, November 1988, or take a look at *Power Graphics Programming*, a collection of "On Graphics" articles.

I hope that helps, Eric. In the next installment we'll get to VGA color selection, which is even more complicated than EGA color selection—but also much more powerful.

IN CONCLUSION

And so, after four installments of intense algorithm development and code optimization, we come to a pause in our explorations of curved objects. Having gotten a solid grasp of circles and ellipses, we're going to spend some time roaming around the VGA before returning for a final aspect of curved objects, the less-than-trivial tilted ellipse.

In response to several requests, the next installment will cover VGA two-stage color lookup hardware, which ultimately gives you your choice of 2, 4, 8, 16 or 256 colors chosen from among 256K colors. The whole process is more than a little strange, owing both to the need for backward compatibility with the EGA and to the unique requirements of 256-color mode, but it's worth it to get such remarkably fine control over the colors your programs display.

After colors, I'm open to suggestions. A recent letter asked if the VGA could be programmed to emulate the Tandy 320x200 16-color and 640x200 4-color modes. I think the answer is yes. The VGA can also be programmed to emulate the Hercules bitmap. Interested? If you are, let me know, and I promise I'll get to it soon.

See you next time. PJ

MISOSYS, Inc.

Programmer's Journal downloads are now available from MISOSYS' forum on CompuServe: PCS49

When you don't have to write in stone, don't let your editor weigh you down. You need SAID-86!

Editing was never so easy!

SAID-86 is a fast, flexible, full screen text editor for PC's. It is perfect for editing batch files, program listings, README files, CONFIG.SYS files, and anything you now do with EDLIN or the non-document mode of a word processor. Why struggle with huge editors; when all is said and done, SAID-86 will be your text editor of choice!

Check out this list of features

- ✓ WordStar-like editing commands are easy to use
- ✓ Pull-down menu system for commanding SAID-86
- ✓ Supports nine editing buffers with automatic swap to disk
- ✓ Supports up to 30 user-defined macros; 255 characters each
- ✓ Undelete the last nine deleted lines can save your bacon
- ✓ MOUSE support with automatic recognition
- ✓ HELP facility; shell to invoke DOS commands from SAID-86
- ✓ SAID-86 can expand or contract TABs

SAID-86 is reasonably priced at just $29.95 + $2S&H

DJ10 Jumbo	$295 +$5S&H
AB10 Tape Adaptor	$75 +$3S&H
KE10 External Adaptor	$110+$5S&H
DC2000 tape	$22.50

M/C & VISA. S&H are U.S. only.

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170-0239
800-MISOSYS
(US&Canada)
or 703-450-4181

Now get the clearest view of your disk drive and memory contents with our DED86™

Powerful features in Version 2!

When you need to travel through your disk drive, why settle for a tool that isolates you from the road? DED86 gives you the direct controls you need to explore your disk. It's a full-screen sector-oriented disk/file editor and a page-oriented memory editor. When you want to "unerase" erased files, DED86's flexible KEEP facility does the job without you fussing over FATs.

Check out this list of features

- ✓ Look by cylinder/head/sector, sector or cluster; even reserved/hidden sectors
- ✓ Scan free clusters to search for erased data ✓ Jump about subdirectories
- ✓ Keep sectors & clusters for writing to a file; Great for partial file recovery
- ✓ Edit bytes in hexadecimal or ASCII, zap in 0s; Undo last edit!
- ✓ Search your disk drive, a file, or memory for ASCII or hexadecimal strings
- ✓ Touch a directory file entry with your date and/or time
- ✓ Obtain complete disk statistics in one screen; includes usage data
- ✓ Alter file attributes: archive, system, hidden, read
- ✓ Save/Restore sectors to/from auxiliary buffers; for moving data around
- ✓ DOS subshell available while using DED86 lets you shell to other programs
- ✓ Handles 5.25" & 3.5" drives & RAM disks; even large partition drives

DED86 is easy on your pocketbook; only $29.95 + $3S&H

Fujitsu Floppy Disk Drives:	5.25" 360K 1/2 height	$75+$4S&H
	3.5" 720K in 5.25" frame	$85+$3S&H
	3.5" 1.44M in 5.25" frame	$90+$3S&H

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    outpw(GC_INDEX, (0xFF << 8) | BIT_MASK_INDEX);
#else
    outp(GC_INDEX, BIT_MASK_INDEX); /* leave the GC Index reg pointing
                                       to the Bit Mask reg */
#endif
    /* Draw the four symmetric arcs for which X advances faster (that is,
       for which X is the major axis) */
    /* Generate the draw list for 1 octant */
    Length = GenerateEOctant(PixList, (long) ASquared * 2 * B,
       (long) ASquared / 4 - ASquared * B, ASquared, BSquared);
    /* Now draw each of 4 symmetries of the octant in turn, the
       octants for which X is the major axis. Adjust every other arc
       so that there's no overlap */
    DrawHOctant(X,Y-B,Length,SCREEN_WIDTH_IN_BYTES,0,PixList);
    DrawHOctant(X+1,Y-B+(*PixList),Length-1,SCREEN_WIDTH_IN_BYTES,1,
       PixList+1);
    DrawHOctant(X,Y+B,Length,-SCREEN_WIDTH_IN_BYTES,0,PixList);
    DrawHOctant(X+1,Y+B-(*PixList),Length-1,-SCREEN_WIDTH_IN_BYTES,1,
       PixList+1);

    /* Draw the four symmetric arcs for which Y advances faster (that is,
       for which Y is the major axis) */
    /* Generate the draw list for 1 octant */
    Length = GenerateEOctant(PixList, (long) BSquared * 2 * A,
       (long) BSquared / 4 - BSquared * A, BSquared, ASquared);

    /* Now draw each of 4 symmetries of the octant in turn, the
       octants for which X is the major axis. Adjust every other arc
       so that there's no overlap */
    DrawVOctant(X-A,Y,Length,-SCREEN_WIDTH_IN_BYTES,1,PixList);
    DrawVOctant(X-A+(*PixList),Y+1,Length-1,SCREEN_WIDTH_IN_BYTES,1,
       PixList+1);
    DrawVOctant(X+A,Y,Length,-SCREEN_WIDTH_IN_BYTES,0,PixList);
    DrawVOctant(X+A-(*PixList),Y+1,Length-1,SCREEN_WIDTH_IN_BYTES,0,
       PixList+1);

#if ISVGA
    /* Restore original write mode */
    outpw(GC_INDEX, (OriginalGCMode << 8) | GC_MODE_INDEX);
    /* Restore normal Color Don't Care setting */
    outpw(GC_INDEX, (0x0F << 8) | COLOR_DONT_CARE);
#else
    /* Reset the Bit Mask register to normal */
    outp(GC_INDEX + 1, 0xFF);
#endif
    /* Turn off set/reset enable */
    outpw(GC_INDEX, (0x00 << 8) | SET_RESET_ENABLE_INDEX);
}
```

LISTING 4

```
; *** (partial version) ***
; <<< This code should replace _GenerateOctant in Listing 4 from   >>>
; <<< "On Graphics" installment #17, PJ 8.2, March 1990, with the  >>>
; <<< resulting code assembled and linked to Listing 3.            >>>
; <<< Alternatively, L4FULL.ASM, available from Programmer's       >>>
; <<< Journal, can be assembled and linked to Listing 3 without    >>>
; <<< modification.                                                >>>
;;*****************************************************************
; Generates an octant of the specified ellipse, placing the results in
; PixList, with a 0 in PixList meaning draw pixel & move only along
; major axis, and a 1 in PixList meaning draw pixel & move along both
; axes.
; C near-callable as:
;  int GenerateEOctant(unsigned char *PixList, long MinorAdjust,
;       long Threshold, long MajorSquared, long MinorSquared);
; Return value = PixelCount (# of points)
; Passed parameters:
GenerateOctantParms     struc
                dw ?            ;pushed BP
                dw ?            ;return address pushed by call
  PixList       dw ?            ;pointer to list to store draw control data in
  MinorAdjust   dd ?            ;initially MajorAxis**2 * 2 * MinorAxis, used
                                ; to adjust threshold after minor axis move
  Threshold     dd ?            ;initially MajorAxis**2 / 4 + MajorAxis**2 *
                                ; MinorAxis, used to determine when to advance
                                ; along the minor axis
  MajorSquared  dd ?            ;MajorAxis**2
  MinorSquared  dd ?            ;MinorAxis**2
GenerateOctantParms     ends
;
; Local variables (offsets relative to BP in stack frame):
PixelCount      equ -2          ;running major axis coordinate
                                ; relative to center
MajorAdjust     equ -6          ;used to adjust threshold after major
                                ; axis move
MajorSquaredTimes2 equ -10      ;MajorSquared * 2
MinorSquaredTimes2 equ  -14     ;MinorSquared * 27
        public _GenerateEOctant
_GenerateEOctant proc   near
        push    bp              ;preserve caller's stack frame
        mov     bp,sp           ;point to our stack frame
        add     sp,MinorSquaredTimes2
                                ;allocate room for local vars
        push    si              ;preserve C register variables
        push    di
;Initialize local variables.
        mov     word ptr [bp+PixelCount],0 ;initialize count of pixels
                                        ; to zero
        mov     ax,word ptr [bp+MajorSquared] ;set MajorSquaredTimes2
        shl     ax,1                    ;lower word times 2
        mov     word ptr [bp+MajorSquaredTimes2],ax
        mov     ax,word ptr [bp+MajorSquared+2]
        rcl     ax,1                    ;upper word times 2
        mov     word ptr [bp+MajorSquaredTimes2+2],ax

        mov     ax,word ptr [bp+MinorSquared] ;set MinorSquaredTimes2
        shl     ax,1                    ;lower word times 2
        mov     word ptr [bp+MinorSquaredTimes2],ax
        mov     ax,word ptr [bp+MinorSquared+2]
        rcl     ax,1                    ;upper word times 2
        mov     word ptr [bp+MinorSquaredTimes2+2],ax
;Set up registers for loop.
        mov     di,[PixList+bp]         ;point DI to PixList
; Set MajorAdjust to 0.
        sub     cx,cx
        mov     si,cx                   ;SI:CX = MajorAdjust

        mov     bx,word ptr [bp+Threshold]      ;DX:BX = threshold
        mov     dx,word ptr [bp+Threshold+2]
; At this point:
;  DX:BX = threshold
;  SI:CX = MajorAdjust
;  DI = PixList pointer
GenLoop:
; Advance the threshold by MajorAdjust + MinorAxis**2.
        add     bx,cx
        adc     dx,si
        add     bx,word ptr [bp+MinorSquared]
        adc     dx,word ptr [bp+MinorSquared+2]
; If the threshold has passed 0, then the minor coordinate has
; advanced more than halfway to the next pixel and it's time to
; advance the minor coordinate by 1 and set the next threshold
; accordingly.
        mov     byte ptr [di],0 ;assume we won't move along the
                                ; minor axis
        js      MoveMajor       ;and, in fact, we won't move minor
; Minor coordinate has advanced.
; Adjust the minor axis adjust value.
        mov     ax,word ptr [bp+MajorSquaredTimes2]
        sub     word ptr [bp+MinorAdjust],ax
        mov     ax,word ptr [bp+MajorSquaredTimes2+2]
        sbb     word ptr [bp+MinorAdjust+2],ax
; Adjust the threshold for the minor axis move
        sub     bx,word ptr [bp+MinorAdjust]
        sbb     dx,word ptr [bp+MinorAdjust+2]
        mov     byte ptr [di],1
MoveMajor:
        inc     di
; Count this point.
        inc     word ptr [bp+PixelCount]
; Adjust the major adjust for the new point.
        add     cx,word ptr [bp+MinorSquaredTimes2]
        adc     si,word ptr [bp+MinorSquaredTimes2+2]
; Stop if the major axis has switched (the arc has passed the
; 45-degree point).
        cmp     si,word ptr [bp+MinorAdjust+2]
        ja      Done
        jb      GenLoop
        cmp     cx,word ptr [bp+MinorAdjust]
        jb      GenLoop
Done:
        mov     ax,[bp+PixelCount]      ;return # of points
        pop     di              ;restore C register variables
        pop     si
        mov     sp,bp           ;deallocate local vars
        pop     bp              ;restore caller's stack frame
        ret
_GenerateEOctant endp
        end
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A four megabyte TSR for the 80386

Running unprotected

By Dean C. Wills

It's easy to be excited yet terrified about beginning software development in a new environment. You look forward to discovering fascinating new features; at the same time you face the uncertainty and challenge of exploring unfamiliar territory. Intel's 80386 protected mode lives up to your expectations on both accounts. It is enticing but terribly complex. It solves all the classic problems which have plagued DOS for years. 80386 protected mode supports segments greater than 64K, bypasses the 640K barrier, and supports virtual memory and multitasking.

But it isn't free! The prices for OS/2 and DOS-extender compilers and debuggers are high. The cost problem multiplies when you consider how difficult it is to lay down a chunk of money for something that you don't even know how to use. Well, never fear. Here I'll solve this problem by introducing you to protected mode at a discount, using standard DOS tools and MS-DOS. All the code for this article was developed using MASM v5.1.

Let me pause a moment here and acknowledge that it is possible to write applications for protected mode without knowing anything about protected mode programming. You simply stick to the applications level and let the operating system manage all the new system features and capabilities. This level of programming can be compared to being a C expert without knowing anything about assembly language or the operating system. Where's the fun in that?

VIRTUAL 8086 MODE

This article focuses on the virtual-8086 (V86) mode of the 80386/486. This subset of protected mode is the best place to begin explaining protected mode programming to DOS programmers because it is the mode capable of running DOS programs unmodified. This mode allows the 80386 to multitask several DOS programs along with other protected mode programs. Each DOS program sits inside a virtual-8086 task which emulates a standard 8086 IBM PC running in non-protected mode.

The DOS program feels that it is running by itself in a completely isolated environment with direct access to all hardware, so that it may perform wanton destruction as it desires. Since this is the way most fast DOS programs run, DOS and V86 mode make a perfect match. V86 mode was designed so well that it is capable of running even the most ill-behaved killer DOS programs imaginable. This mode, I believe, will guarantee the ultimate success of the 386 and its successors; it preserves the valuable DOS investment yet provides all the features for developing powerful 386 protected mode programs.

A V86 MONITOR

The code provided with this article is for an elementary V86 monitor which will put DOS inside a virtual-8086 machine and perform some basic emulation. The program will place the computer in protected mode, create a virtual-8086 task, return to DOS inside of the virtual-8086 task, and execute a DOS terminate and stay resident call. In virtual 8086 mode, all of your familiar DOS programs will still run, and almost as quickly (though note that some instructions that load segment registers or call software interrupts actually take longer in protected mode). But the computer will be in protected mode, and this will open up the capabilities of the machine. This TSR can now be expanded using extended memory to create programs beyond the capability of the 640K DOS machine. Multitasking software, four-megabyte printer buffers, debuggers and LIM expanded memory emulators are just a few of the possibilities.

But be careful! Systems programming on the 386 will probably be the most challenging programming you've ever attempted. Systems assembly language is very different from application assembly language. The instructions you use control the operation of the chip itself. Debugging becomes a nightmare. Standard debuggers are useless because systems

Dean C. Wills is a programmer, attorney, and president of Sandpaper Software, which specializes in 386 programming. He can be reached via FAX: (818) 705-2567.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

programs are written at a lower level than the debuggers themselves. The only way to tell if the program runs is to try it and see what happens.

If it doesn't work, you probably won't be able to examine memory to get an idea of what went wrong. When a systems program goes awry the computer often hangs or resets itself. The best technique that I know of for surviving systems programming is to proceed with caution and make small, well-thought-out modifications to known good software. You have an advantage now because the sample code provided with this article can be your first level of "known good" software.

REAL VERSUS PROTECTED MODE

The primary difference between protected mode and real mode programming is in the use of the segment registers. In both modes, linear addresses are calculated by adding an offset to a base which is generated by the segment registers. In real and virtual-8086 modes, the CPU generates the base address by multiplying the segment register value by 16. In protected mode, on the other hand, segment registers become selectors that refer to a descriptor table entry which contains the base address corresponding to each selector. In protected mode, all memory access is done via these descriptor table entries. The descriptors also contain other information about the segment including its type (executable, read only, or read/write) and length.

There are three system descriptor tables:

- GDT—the global descriptor table
- LDT—the local descriptor table
- IDT—the interrupt descriptor table

The GDT contains a list of descriptors which are shared by all tasks. Whether these selectors can be used by a task is another matter entirely, determined by the privilege level of the task. There are four privilege levels ranging from 0 (highest) to 3 (lowest). Attempted accesses to higher privilege system resources generate GP or General Protection faults. Virtual-8086 mode is fixed at the bottom of the totem pole with privilege level 3.

The LDT is optional and will normally exist for each task only if it is desirable to completely isolate one task from another. Since our example V86 monitor will currently be executing only one task at a time, we won't use a separate LDT.

THE INTERRUPT TABLE

Faults generate interrupts which are defined in the IDT. This table is also used for hardware interrupts and for the INT *n* software instructions. There are 32 possible faults or exceptions, only some of which are defined on the 386. These correspond to interrupts 0 through 31. Faults and exceptions are not new with protected mode. The original 8086 contained a few faults (pun intended) and a warning that "interrupts 0 through 31 were reserved for future processors." Interrupt 0, for example, is the divide fault.

Since the warning was not heeded by the designers of the PC, several of the interrupt faults new with the 286 and 386

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
TITLE   V86 Monitor
;
;Copyright 1990 Sandpaper Software by Dean C. Wills
;
;art.exe:    art.asm
;    masm art,art,art/MX;
;    link art /MA/LI/NOE;
;
.MODEL  MEDIUM,C
.386P
;
;Define the Data Segment, DGROUP
;
_DATA           SEGMENT WORD USE16 PUBLIC 'DATA'
_DATA           ENDS
CONST           SEGMENT WORD USE16 PUBLIC 'CONST'
CONST           ENDS
BSS             SEGMENT WORD USE16 PUBLIC 'BSS'
BSS             ENDS
DGROUP_         SEGMENT WORD USE16 PUBLIC 'LAST'
DGROUP_end      LABEL   BYTE            ;the end of the DGROUP segment
DGROUP_         ENDS
DGROUP  GROUP   _DATA, CONST, BSS, DGROUP_
ASSUME          ds:DGROUP
;*************************************
ITEXT           SEGMENT PARA USE16 PUBLIC 'CODE'
ITEXT           ENDS
ITEXT_          SEGMENT PARA USE16 PUBLIC 'CODE'
ITEXT_end       LABEL   BYTE            ;the end of the ITEXT segment
ITEXT_          ENDS
;*************************************
ASSUME          ds:IDATA, ss:IDATA
IDATA           SEGMENT WORD USE16 STACK
MsgNoVM         DB      '* Unable to enter protected mode$'
                DW      100h DUP (?)
ustack          LABEL   WORD            ;init/user stack
IDATA           ENDS
IDATA_          SEGMENT WORD USE16
IDATA_end       LABEL   BYTE            ;the end of the IDATA segment
IDATA_          ENDS
;*************************************
_TEXT           SEGMENT PARA USE16 PUBLIC 'CODE'
_TEXT           ENDS
_TEXT_          SEGMENT PARA USE16 PUBLIC 'CODE'
_TEXT_end       LABEL   BYTE            ;the end of the _TEXT segment
_TEXT_          ENDS
;*************************************
_DATA           SEGMENT WORD USE16 PUBLIC 'DATA'
;-------Types of Descriptors    - SYSTEM
TSS             EQU     1               ;286 Task State Segment
LDT             EQU     2               ;Local Descriptor Table
TSS_BUSY        EQU     3               ;286 TSS, current or back-linked
CALL_GATE       EQU     4               ;286 Call Gate
TASK_GATE       EQU     5               ;Task Gate
INT_GATE        EQU     6               ;286 Int Gate, int proc, clr ints
TRAP_GATE       EQU     7               ;286 Same as Int Gate don't clr ints
XTSS            EQU     9               ;386 Task State Segment
XTSS_BUSY       EQU     0bh             ;386 TSS, current or back-linked
XCALL_GATE      EQU     0ch             ;386 Call Gate
XINT_GATE       EQU     0eh             ;386 Int Gate, int proc, clr ints
XTRAP_GATE      EQU     0fh             ;386 Same as Int Gate don't clr ints
;-------Types of Descriptors    - MEMORY
RDONLY          EQU     10h             ;read only
RD_WR           EQU     12h             ;read/write
RD_WR_XD        EQU     16h             ;read/write expand down
EXONLY          EQU     18h             ;execute only
EX_RD           EQU     1Ah             ;execute/readable
EXONLY_CF       EQU     1Ch             ;execute only/conforming
EX_RD_CF        EQU     1Eh             ;execute/readable/conforming
;---------------------------------------
GDLIMIT MACRO   granularity,default,highlimit
        DB      (granularity SHL 7) OR (default SHL 6) OR (highlimit AND 0fh)
        ENDM
PLTYPE  MACRO   privilege,type
        DB      80h OR (privilege SHL 5) OR TYPE
        ENDM
;---------------------------------------
ASSUME  ds:DGROUP
ALIGN   16
gdt     LABEL   BYTE                    ;GLOBAL DESCRIPTOR TABLE
        DQ      0                       ;00 GDT(0) always blank
        DW      (GDTSIZE SHL 3) - 1     ;08 DATA - GDT  limit 15..0
        DW      OFFSET DGROUP:gdt       ;base 15..0
GDLIMIT 0,0,0                           ;granularity/default/limit 19..16
PLTYPE  0,RD_WR                         ;pl/type
        DW      DGROUP                  ;segment (added to base)
        DW      (IDTSIZE SHL 3) - 1     ;10 DATA - IDT  limit 15..0
        DW      OFFSET DGROUP:idt       ;base 15..0
GDLIMIT 0,0,0                           ;granularity/default/limit 19..16
PLTYPE  0,RD_WR                         ;pl/type
        DW      DGROUP                  ;segment (added to base)
x_DGROUP  = $-gdt                       ;18 DATA - ds of DGROUP segment
        DW      (OFFSET DGROUP:DGROUP_end)-1    ;limit 15..0
        DW      0                       ;base 15..0
GDLIMIT 0,0,0                           ;granularity/default/limit 19..16
PLTYPE  0,RD_WR                         ;pl/type
        DW      DGROUP                  ;segment (added to base)
        DW      (OFFSET DGROUP:DGROUP_end)-1    ;20 DATA - es  limit 15..0
        DW      0                       ;base 15..0
GDLIMIT 0,0,0                           ;granularity/default/limit 19..16
PLTYPE  0,RD_WR                         ;pl/type
        DW      DGROUP                  ;segment (added to base)
        DW      (OFFSET IDATA:IDATA_end)-1      ;28 DATA - ss  limit 15..0
        DW      0                       ;base 15..0
GDLIMIT 0,0,0                           ;granularity/default/limit 19..16
PLTYPE  0,RD_WR                         ;pl/type
        DW      IDATA                   ;segment (added to base)
        DW      (OFFSET ITEXT:ITEXT_end)-1      ;30 CODE - cs  limit 15..0
        DW      0                       ;base 15..0
GDLIMIT 0,0,0                           ;granularity/default/limit 19..16
PLTYPE  0,EXONLY                        ;pl/type
        DW      ITEXT                   ;segment (added to base)
        DQ      0                       ;38 (0)  - int 15h reserved
x_CRT  = $-gdt                          ;40 DATA - video ram  descriptor of
CRT segment
        DW      0fffffh                 ;limit 15..0
        DW      0                       ;base 15..0
GDLIMIT 0,0,0                           ;granularity/default/limit 19..16
PLTYPE  0,RD_WR                         ;pl/type
        DW      0b800h                  ;segment (added to base)
x_TEXT  = $-gdt                         ;48 CODE - cs  descriptor of _TEXT
segment
        DW      (OFFSET _TEXT:_TEXT_end)-1      ;limit 15..0
        DW      0                       ;base 15..0
GDLIMIT 0,0,0                           ;granularity/default/limit 19..16
PLTYPE  0,EX_RD                         ;pl/type
        DW      _TEXT                   ;segment (added to base)
x_TSS   = $-gdt                         ;50 DATA - Tss of main task
        DW      OFFSET DGROUP:task_end-OFFSET DGROUP:task-1      ;limit
        DW      OFFSET DGROUP:task      ;base 15..0
GDLIMIT 0,0,0                           ;granularity/default/limit 19..16
PLTYPE  0,XTSS                          ;pl/type
        DW      DGROUP                  ;segment (added to base)
x_PC   = $-gdt                          ;58h, PC address space
        DW      0fffffh                 ;
        DW      0                       ;base 15..0
GDLIMIT 1,0,0fh                         ;granularity/default/limit 19..16
PLTYPE  0,RD_WR                         ;pl/type
        DW      0                       ;segment
;---------------------------------------
GDTSIZE =       (($-gdt) SHR 3)         ;number of entries in gdt
ALIGN   16
idt     LABEL   BYTE                    ;INTERRUPT DESCRIPTOR TABLE
IRP     z,,01,02,03,04,05,06,07,08,09,10,11,12,13,14,??,16
        DW      OFFSET _TEXT:fault&z, x_TEXT,(XINT_GATE OR 80h) SHL 8,0
ENDM
;          17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
IRP     z,??,??,??,??,??,??,??,??,??,??,??,??,??,??,??,??
        DW      OFFSET _TEXT:fault&z, x_TEXT,(XINT_GATE OR 80h) SHL 8,0
ENDM
IRP     z,,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f
        DW      OFFSET _TEXT:inthw&z, x_TEXT,(XINT_GATE OR 80h) SHL 8,0 ;hw
ENDM
;---------------------------------------
IDTSIZE =       (($-idt) SHR 3)         ;number of entries in idt
;---------------------------------------
task    DB      68h DUP (?)             ;uninitialized tss block
iobase  DW      OFFSET virtual_io - OFFSET task ;I/O MAP BASE
virtual_io  DB  256 DUP (0)             ;all I/O is allowed
        DD      -1                      ;extra dword
task_end        LABEL   BYTE
;---------------------------------------
        DW      512 DUP (?)             ;stack for level 0 execution
stack0  LABEL   WORD
cursor  LABEL   WORD                    ;cursor for screen
cury    DB      0                       ;column
curx    DB      0                       ;row
fault   DB      "Fault/Exception #",0
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

V86 mode was designed so well that it is capable of running even the most ill-behaved killer DOS programs imaginable.

conflict with the usage on the PC. For example, both the GP fault and the LPT2's IRQ5 generate INT 0Dh. The numeric coprocessor fault #16 conflicts with the INT 10h video ROM BIOS call. In addition, DOS requires that the interrupts be serviced by the routines listed in the interrupt vector table located at the start of memory.

In our V86 monitor example, the solution to these problems has been handled as follows. First of all, the 8259s are reprogrammed so that the hardware interrupts occur at interrupts 20h through 27h for the first chip and 28h through 2Fh for the second chip. Set this way, the hardware interrupts won't conflict with the exceptions and faults. Each of the descriptors in the IDT is then set up with privilege level 0.

This means that if a program running in virtual-8086 mode attempts to call the video BIOS via INT 10h, a GP exception will occur because the privilege level of V86 mode is fixed at 3. At that point, the GP exception handler will emulate the software interrupt and pass control to the routine listed in the interrupt vector table. Hardware interrupts will emulate the old DOS vectors in the same manner.

OTHER DESCRIPTOR TYPES

A number of different types of descriptors can be contained in the descriptor tables. The two basic types are segment descriptors which indicate the base, length, and type of a memory area, and a gate descriptor which is an indirect jump to another location. The GDT table will be made up of the necessary segment descriptors, whereas the IDT will contain gate descriptors pointing to the addresses of the fault, exception, and hardware interrupt routines.

The formats of these two descriptors are different. The gate descriptors are easily created by the MASM **DW** directive. The segment descriptors are convoluted and are most easily created using a software routine that adjusts the data into its proper format. This is done by the **adjust** subroutine. This subroutine adjusts all segment descriptors except for gate descriptors, which are already in the proper format.

TO THE PROTECTED MODE

Once the GDT and IDT are adjusted, the move to protected mode is made using the INT 15h service AH=89h routine (switch processor to protected mode). This routine also reprograms the 8259 interrupt controllers to use 20h and 28h as the base addresses for hardware interrupts.

Once you are in protected mode, a far jump should be made to clean out the prefetch queue which was read while the processor was still in real mode. In addition, all the segment registers must be reset to protected mode descriptors or to 0. In the example code, these steps are not explicitly necessary because they are taken care of inside the INT 15h ROM BIOS routine.

Each task has a task state segment (TSS) which defines the parameters for the task. This is used when multitasking so that tasks can call one another, and all the information pertaining to that task can be saved somewhere. The definition of the TSS is a descriptor itself, and it is defined in the GDT and loaded into the task register once we are in protected mode. The TSS of a task contains, among other things, the descriptor for the LDT of the task. This is loaded to become the current LDT when the task is made active. The TSS must therefore be defined in the GDT, for if it were defined in the LDT it would end up redefining itself when the LDT is loaded during task activation.

After the move to the protected mode is complete, we go about initializing the segment registers and the protected mode data structures, such as our TSS. We also have to reinitialize all the segment registers whenever we go from V86/real mode to protected mode or vice-versa. This is because segment values (selectors) defined in one mode turn out to be meaningless in the other mode. For example, the segment of color video RAM in V86/real mode is B800h; in the protected mode the selector is 40h.

Setting up the TSS is straightforward. We don't need to fill every element of the TSS because some of these are automatically set by the CPU during task transitions. Others, such as the LDT page register and the stack to use while in the protected mode, are static and must be defined in advance.

THROUGH THE GATE

Now comes the tricky part. We pretend as if we entered protected mode via an int_gate called from a V86 program instead of by INT 15h. We do this by pushing the necessary V86 register values on the stack in the correct order. We push old style (base*16) segment values on the stack. The CS and IP are defined to point at the exit to DOS routine. The flag register on the stack has Virtual Mode (bit 17) set so that after the return the processor will execute in V86 mode. The flag register on the stack must also be set to say that IOPL is level 3 for this task or GP exceptions will be generated on the **popf**, **iret**, **cli**, **sti** and **pushf** instructions.

This feature allows the processor the ability to prevent one task from clearing interrupts and dying, taking all of the other active tasks with it. If the Nested task (bit 14) is set in the current flags, the processor will attempt to return to a backlinked task instead of going to the stack for the **iretd**. We clear the NT bit and the **iretd** will pop all the values off the stack and change to V86 mode.

In V86 mode, the program calls DOS to terminate and stay resident. The software will run in V86 mode until a hardware

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

or software interrupt or exception occur. Either of these will cause an int_gate transition through the IDT to the interrupt service routine. An int_gate transition also clears interrupts so it won't be necessary to worry about hardware interrupts occurring during protected mode processing. There are three important interrupt processing routines. The first is **intvm**, which handles the hardware interrupts; next is **fault13**, which handles the general protection exception caused by the software interrupts; and the third is **fail**, which handles bugs or shortcomings in the V86 monitor itself.

HANDLING INTERRUPTS

The hardware and software interrupt handlers are almost identical. They both determine which interrupt should be issued to the V86 program, and pass this interrupt number to the **emulateint** routine. This routine saves the V86 code segment and instruction pointer along with the flags on the V86 stack. It also sets the V86 code segment and instruction pointer equal to the address from the V86 interrupt vector table at the start of system memory. This process parallels the actual interrupting mechanism of real mode.

The GP exception handler has a bit of translation to do before it emulates the software interrupt. It first must access the user memory using protected mode descriptors to determine which INT *n* instruction was attempted. This is done by having a descriptor defined which is capable of accessing the entire four-gigabyte address space with 32-bit offsets. The V86 code segment and instruction pointer, pushed on the stack by the **int_gate** transition, are combined using the base*16+offset method to generate a 32-bit offset. This offset points at the faulting instruction. This instruction is verified to make sure that it is an INT *n* instruction. If it is, the vector number is retrieved from memory and the interrupt is emulated.

The hardware interrupt handlers are simpler. They undo the translation via **intvm** which is now being done by the 8259 and refer the hardware interrupt back to the standard DOS vector number. Other faults and exceptions also use **intvm** to send the interrupt to the V86 interrupt handler. Interrupts which cannot be handled in this way are passed to the **fail** routine. The **fail** routine places an error message and stack dump on the screen and terminates the program.

IN CONCLUSION

That completes the explanation of the software. I've left a few details out which cloud the general picture, yet which are easily understood through examination of the source code. The code is defined to work with a color monitor; a small modification of the GDT can be made to convert it to monochrome. The code doesn't detect whether you have an 80386 chip. If you want to add this, see "386 Now" by Steve Baker in *PJ* 7.2 (March/April 1989).

I hope this will get you going in 386 protected mode. By now you probably realize that an 80386 programmer's reference manual would be a good investment. When you modify the software, leave a trail of successive backups and be meticulous in the modifications, or you just might get lost in the protected mode! PJ

```
buffer  DB      10 DUP (?)          ;buffer for display number
hex_xlt DB '0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'
stepflag  DW    0                   ;flag for single step
_DATA           ENDS
;*************************************
_TEXT           SEGMENT PARA USE16 PUBLIC 'CODE'
ASSUME  cs:_TEXT, ds:nothing, es:nothing, fs:nothing, gs:nothing,
ss:DGROUP
dataseg DW      x_DGROUP
crtseg  DW      x_CRT
pcseg   DW      x_PC
;-------------------------------------
;EQUATES for emulateint, fault13, and intvm
Zgs     EQU     [bp+40]             ;V86 gs from int_gate
Zfs     EQU     [bp+36]             ;V86 fs from int_gate
Zds     EQU     [bp+32]             ;V86 ds from int_gate
Zes     EQU     [bp+28]             ;V86 es from int_gate
Zss     EQU     [bp+24]             ;V86 stack segment from int_gate
Zsp     EQU     [bp+20]             ;V86 stack pointer from int_gate
Zflags  EQU     [bp+16]             ;V86 flags from int_gate
Zcs     EQU     [bp+12]             ;V86 code segment from int_gate
Zip     EQU     [bp+8]              ;V86 instruction pointer...
Zdword  EQU     [bp+4]              ;error code-fault13 or vector-intvm
Zebp    EQU     [bp]                ;saved ebp on stack
Zeax    EQU     [bp-4]              ;saved eax on stack
Zebx    EQU     [bp-8]              ;saved ebx on stack
;-------------------------------------
;Macro for faults, exceptions, and interrupts
; Fault and exception macro
fejmp macro lbl, nptr, jump
fault&lbl:      push    dword ptr nptr
                jmp     jump
                endm
; Hardware interrupt macro
hijmp macro lbl, nptr, jump
inthw&lbl:      push    dword ptr nptr
                jmp     jump
endm
;emulateint() does the job of emulating the interrupt. It pulls the DOS
;vector from the int vector table. Stack: Int_gate, dword, ebp, eax,
;ebx. ebx=vector
emulateint:
        sub     eax,eax             ;clear eax for SIB, calculation later
        mov     ebx,fs:[ebx*4+eax]  ;retrieve return vector
        mov     ax,bx               ;save the offset to ax
        shr     ebx,16              ;calculate segment
        xchg    bx,[Zcs]            ;get current segment,save new segment
        xchg    ax,[Zip]            ;get current offset, save new offset
        shl     ebx,16              ;
        mov     bx,ax               ;ebx=segment:offset
        sub     word ptr [Zsp],6    ;make room on stack
        mov     ax,[Zss]            ;stack segment(high eax already zero)
        shl     eax,4               ;convert from paragraphs
        add     eax,[Zsp]           ;stack pointer
        mov     fs:[eax],ebx        ;segment:offset
        mov     bx,[Zflags]         ;get current flags
        mov     fs:[eax+4],bx       ;save them on stack. clr TF, RF, IF
        and dword ptr [Zflags],NOT ((1 SHL 8) OR (1 SHL 9) OR (1 SHL 16))
        pop     ebx                 ;
        pop     eax                 ;
        pop     ebp                 ;
        add     esp,4               ;kill error code or vector
        btr     [stepflag],0        ;single step needed? Only via fault13
        jc      fault01             ;yes, do it with only int_gate stack
        iretd
;
;intvm issues an int to the user. It first checks to see if we were in
;PM. This is possible only for faults 0 through 04 because interrupts
;are left CLEAR while in protected mode. Inputs: An int-gate transition
;with the vector number on the stack. We use a simular stack frame for
;intvm, emulateint, and fault13. Intgate, Dword, ebp, eax, ebx, ebp
;points at the dword.
intvm:  test    BYTE PTR [esp+12+2],1 SHL 1  ;Test flags: Coming from VM?
        jz      fail                ;we're in PM! (must be 0 through 4)
        push    ebp                 ;for use here
        mov     ebp,esp             ;for our stack frame(easier than esp)
        push    eax                 ;for use here
        push    ebx                 ;for use here
        mov     ebx,[Zdword]        ;get the vector number from the stack
        mov     fs,[pcseg]          ;pc address space
        jmp     emulateint          ;emulate the interrupt
;
;fault13 is the general protection fault handler for virtual 86 mode
fault13:test    BYTE PTR [esp+12+2],1 SHL 1     ;test VM/NOT
        jz      faultgp             ;protected mode gp
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        push    ebp                     ;save ebp
        mov     ebp,esp                 ;
        push    eax                     ;
        push    ebx                     ;save eax & ebx
        mov     fs,[pcseg]              ;the PC address space
        movzx   eax,word ptr [Zcs]      ;code segment where executing
        shl     eax,4                   ;calculate base * 16
        add     eax,[Zip]               ;add IP offset to point at instruction
        movzx   ebx,word ptr fs:[eax]   ;get the instruction (two bytes)
        cmp     bl,0cdh                 ;int_n instruction?
        jnz     $other                  ;no, something else
        shr     ebx,8                   ;slide out 0xcd, put vector in bl
        add     word ptr [Zip],2        ;skip int_n instruction
        btr     word ptr [Zflags],8     ;clear Trap flag
        setc    byte ptr [stepflag]     ;If it was set, we need single step
        jmp     emulateint              ;emulate it, ebx=vector
;$other handles non int_n instructions such as int3/into
;It may also need to handle out/in/popf/sti,...
;Right now, it just generates a #GP() error.
$other: pop     ebx                     ;
        pop     eax                     ;
        pop     ebp                     ;clean off stack (except error code)
        jmp     faultgp                 ;handle it the normal way
;---------------------------------------
;Faults and Exceptions other than GP (13). Since 00 through 02 are supported
;by the BIOS they are redirected there. Ints 3 and 4 are also supported by
;the BIOS but they won't come here. They'll go to the fault13 handler for
;emulation, but fault13 doesn't support them yet. Here, we say fail
;
fejmp   00,0,intvm                      ;Divide fault
fejmp   01,1,intvm                      ;Debug exception
fejmp   02,2,intvm                      ;NMI
fejmp   03,3,fail                       ;Breakpoint
fejmp   04,4,fail                       ;Interrupt on overflow
fejmp   05,5,fail                       ;Bounds check
fejmp   06,6,fail                       ;Invalid opcode
fejmp   07,7,fail                       ;Coprocessor not available
fejmp   08,8,failec                     ;Double fault
fejmp   09,9,fail                       ;Coprocessor segment overrun
fejmp   10,10,failec                    ;Invalid tss
fejmp   11,11,failec                    ;Segment not present
fejmp   12,12,failec                    ;Stack exception
fejmp   gp,13,failec                    ;General Protection Fault
fejmp   14,14,failec                    ;Page fault
fejmp   16,16,fail                      ;Coprocessor error
fejmp   ??,32,fail                      ;Unknown Exception

;---------------------------------------
;Hardware Interrupts
hijmp   00,08h,intvm                    ;issue int for timer
hijmp   01,09h,intvm                    ;issue int for keyboard
hijmp   02,0ah,intvm                    ;issue int for slave
hijmp   03,0bh,intvm                    ;issue int for com2
hijmp   04,0ch,intvm                    ;issue int for com1
hijmp   05,0dh,intvm                    ;issue int for lpt2
hijmp   06,0eh,intvm                    ;issue int for floppy
hijmp   07,0fh,intvm                    ;issue int for lpt1
hijmp   08,70h,intvm                    ;issue int for rtc
hijmp   09,71h,intvm                    ;issue int for redirected irq02
hijmp   0a,72h,intvm                    ;issue int extra
hijmp   0b,73h,intvm                    ;issue int extra
hijmp   0c,74h,intvm                    ;issue int extra
hijmp   0d,75h,intvm                    ;issue int for math coprocessor
hijmp   0e,76h,intvm                    ;issue int for fixed disk
hijmp   0f,77h,intvm                    ;issue int extra
;---------------------------------------
;failec() fails with an error code. It combines the error code and
;faultnumber into a single dword and sets (1 SHL 15) as a flag.
;fail() fails with the fault number on the stack. It's job is to display
;information on the screen. For some faults (stack, double, Invalid TSS)
;this is going to fail and hang the system. The int_gate for these fault
;should be instead a TSS for this to work.
;
failec: xchg    [esp+4],eax             ;save eax, fetch error code
        shl     eax,16                  ;move it to high eax
        or      eax,1 SHL 15            ;set bit for error code
        or      ax,word ptr [esp]       ;OR in fault number
        xchg    [esp+4],eax             ;restore eax, save errorcode/fault#
        add     esp,4                   ;kill fault # on stack.
                                        ;EC/fault still
fail:   xchg    [esp],eax               ;get error code, save eax
        pushad                          ;save the registers we will display
        push    ds                      ;
        push    es                      ;
        mov     ebp,esp                 ;stack pointer
        mov     ds,[dataseg]            ;DGROUP selector
        mov     es,[dataseg]            ;DGROUP selector
ASSUME  ds:DGROUP, es:DGROUP
        mov     si,OFFSET DGROUP:fault  ;fault message
        push    eax                     ;save error code/fault number
        push    ax                      ;save fault number alone
        call    showmsg                 ;show the line -int number
        pop     ax                      ;restore fault number
        aam                             ;convert al to decimal
        add     ax,('0' SHL 8)+'0'      ;convert to ascii
        xchg    al,ah                   ;
        call    showchar                ;tens digit
        xchg    al,ah                   ;
        call    showchar                ;ones digit
        mov     al,' '                  ;
        call    showchar                ;
        pop     eax                     ;restore error code/fault #
        test    ax,1 SHL 15             ;is there an error code?
        jz      @F                      ;
        mov     al,'('                  ;Display Error Code
        call    showchar                ;
        shr     eax,16                  ;now the error code in ax
        call    showwordh               ;show word in hex
        mov     al,')'                  ;show parenthesis
        call    showchar                ;
        mov     al,' '                  ;
        call    showchar                ;@@ in MASM 5.1 is a local label
@@:     mov     al,'@'                  ;Show fault-address
        call    showchar                ;
        mov     ax,[bp+44]              ;cs
        call    showwordh               ;show word in hex
        mov     al,':'                  ;
        call    showchar                ;
        mov     ax,[bp+40]              ;ip
        call    showwordh               ;show word in hex
        mov     al,' '                  ;spacer
        call    showchar                ;
        mov     cx,40                   ;dump the last 40 words of memory
        mov     si,bp                   ;top of the stack
@@:     lodsw                           ;get a word from the stack
        call    showwordh               ;show word in hex
        mov     al,':'                  ;spacer
        call    showchar                ;
        loop    @B                      ;loop for each word
@@:     jmp     @B                      ;IDLE loop!
;---------------------------------------
;showwordh() shows a word in hex.
showwordh:
        push    si                      ;save si
        mov     dx,ax                   ;into dx for display
        mov     di,OFFSET DGROUP:buffer ;point at the buffer
        mov     si,di
        mov     bx,OFFSET DGROUP:hex_xlt;point at translation table
        call    wordh                   ;place the word in the buffer
        mov     byte ptr es:[di],0      ;terminate the string
        call    showmsg                 ;display the string
        pop     si                      ;restore si
        ret

wordh:  mov     al,dh                   ;first the high byte
        call    byteh                   ;
        mov     al,dl                   ;now the low byte
byteh:  push    ax                      ;
        shr     al,4                    ;
        xlat                            ;convert binary to ascii via table
        stosb                           ;
        pop     ax                      ;
        and     al,0fh                  ;
        xlat                            ;convert binary to ascii via table
        stosb                           ;
        ret                             ;
;
;showchar() displays an ascii character
assume  ds:DGROUP, es:nothing
showchar:
        push    ax                      ;
        push    es                      ;
        call    getpos                  ;get video location to es:di
        mov     ah,7                    ;normal attribute
        stosw                           ;
        call    setpos                  ;set new cursor position
        pop     es                      ;from es:di
        pop     ax                      ;
        ret
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TOMORROW'S SOFTWARE FOR TODAY'S APPLICATIONS

Quintus' premier Prolog-based software engineering products provide methods which enhance software development and maintaince, and offer rapid prototyping, debugging, windowing and database interface capabilities.

Quintus proudly offers support for:

UNIX	Quintus Prolog, Runtime Generator, Runtime License Licenses, Quintus ProWINDOWS, Database Interfaces
MAC	*MacPROLOG, MacOBJECT, Prolog ++, flex
MS-DOS	*DOSProlog, Prolog ++, flex

Now available across a wide variety of platforms including SUN, Digital VAX/VMS, VAX Ultrix and DECstation, HP, Sony, Apollo Domain, IBM RT/PC (AIX) and IBM PS/2 (AIX), 80386 UNIX, Sequent Symmetry and Intergraph Clipper.

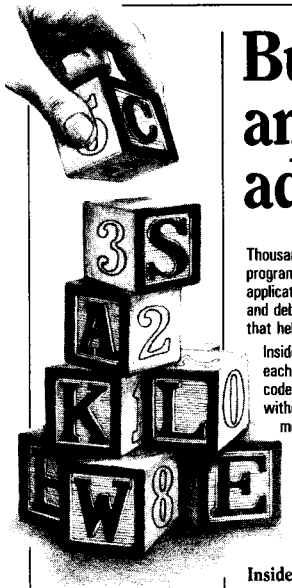
For more information contact sales:

Quintus Computer Systems Inc.
1310 Villa Street, Mt. View, CA 94041 Phone: (800) 542-1283
In California call: (415) 965-7700 FAX: (415) 965-0551

Quintus
An Intergraph Company

The companies and products mentioned are trademarks and registered trademarks of their respective companies.
*Registered trademark of Logic Programing Associates

Circle **19** on Reader Service Card



Build with an Inside! advantage...

Thousands of C, BASIC, FORTRAN and Modula-2 programmers are already using Inside! to improve application performance. You have the best compiler and debugger, now add the performance analysis tool that helps create world class applications!

Inside! analyzes your program in real-time, timing each instance of a function, source line or arbitrary code fragment, all with microsecond accuracy and without source code modification. Utilize the following modes to know where to focus your optimization efforts for maximum reward.

- Function/procedure timing
- Call target timing
- Source Line timing
- Arbitrary event timing
- Non-executed functions or source lines
- MS-DOS function timing
- Overlay and assembly language support

Inside! is available for popular compilers . . .

- Turbo C
- Turbo Pascal
- Microsoft C/QuickC
- Microsoft QuickBASIC
- Microsoft FORTRAN
- Logitech Modula-2
- JPI TopSpeed Modula-2
- WATCOM C
- Zortech C++

Call or write today for a free brochure and the latest list of Inside! products.

To Place Orders
(800)537-5043
Product Support
(508)478-0499
Visa/Mastercard/C.O.D. Accepted

$125.00 EACH
Satisfaction guaranteed

Paradigm Systems
P.O. Box 152 Milford, Massachusetts 01757
Inside! is a trademark of Paradigm Systems.

PARADIGM SYSTEMS INCORPORATED

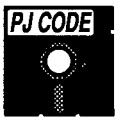
Contact this advertiser directly.


```
;showmsg displays a message on the screen at the current cursor position.
assume  ds:DGROUP, es:nothing
showmsg:push    es
        call    getpos              ;get video location to es:di, dirty ax
        mov     ah,7                ;normal attribute
        jmp     short $shst         ;start
@@:     stosw                       ;
$shst:  lodsb                       ;
        or      al,al               ;at the end?
        jnz     @B                  ;
        call    setpos              ;set new cursor position from
es:di
        pop     es                  ;
        ret
;-------------------------------------
;getpos gets the location of the video cursor into es:di
assume  ds:DGROUP, es:nothing
getpos:
        push    ax
        mov     es,[crtseg]         ;segment of video ram
        mov     al,[cury]           ;cursor y
        mov     ah,80*2             ;bytes/line
        mul     ah                  ;ax is the start of line [cury]
        mov     di,ax               ;
        movzx   ax,[curx]           ;x offset
        shl     ax,1                ;times 2 (attribute/char)
        add     di,ax               ;di is the offset now
        pop     ax
        ret
;-------------------------------------
;setpos sets the cursor from es:di
assume  ds:DGROUP, es:nothing
setpos: push    cx
        shr     di,1                ;to characters
        mov     ax,di               ;
        mov     cl,80               ;characters/line
        div     cl                  ;
        pop     cx                  ;
        mov     [cursor],ax         ;save new cursor
        ret
;-------------------------------------
_TEXT           ENDS
;*************************************
ITEXT           SEGMENT PARA USE16 PUBLIC 'CODE'
;
;start() is the main entrypoint to the software
assume  cs:ITEXT, ds:nothing, es:nothing, ss:IDATA
start:  mov     ax,3                ;clear screen
        int     10h                 ;
        mov     ax,DGROUP           ;set the segment to use.
        mov     ds,ax               ;
        mov     es,ax               ;
ASSUME  ds:DGROUP, es:DGROUP
        mov     si,OFFSET idt       ;point at the idt
        mov     cx,IDTSIZE          ;number of entries
        call    adjust              ;adjust descriptor tables
        mov     si,OFFSET gdt       ;point at the gdt
        mov     cx,GDTSIZE          ;number of entries
        call    adjust              ;adjust descriptor tables
        mov     si,OFFSET gdt       ;point at the gdt
        mov     bh,20h              ;int level 1 start
        mov     bl,28h              ;int level 2 start
        mov     ah,89h              ;code for enter protected mode
        mov     cx,0FFFFh           ;idle here to ensure all
        loop    $                   ;DOS keybd ints processed
ready:  int     15h                 ;BIOS call
        jc      failpm              ;couldn't enter pm
assume  cs:ITEXT,ss:nothing,ds:nothing,es:nothing,fs:nothing,gs:nothing
        sub     ebx,ebx             ;high stack is 0 for [esp] access
        mov     bx,OFFSET DGROUP:stack0 ;stack for privilege 0 execution
        mov     ax,x_DGROUP         ;dgroup selector
        mov     ss,ax               ;stack
        mov     esp,ebx             ;set stack(32 bit reg, 16 bit seg)
        mov     ds,ax               ;fs, gs are legal by int 15h
assume  ss:DGROUP, ds:DGROUP        ;
        mov     ax,x_TSS            ;set tss for this task
        ltr     ax                  ;set task register
        sub     ax,ax               ;ldt0
        lldt    ax                  ;no ldt
        mov     word ptr [task+08h],ss  ;save level 0 stack segment in TSS
        mov     word ptr [task+04h],sp  ;save level 0 stack pointer in TSS
        mov     ax,[iobase]         ;I/O base for V86 task
        mov     word ptr [task+66h],ax  ;store I/O base address in TSS
        sub     eax,eax             ;clear high part of eax
        mov     ax,IDATA            ;initial data segment of V86 task
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        push    eax                     ;gs
        push    eax                     ;fs
        push    eax                     ;ds
        push    eax                     ;es
        push    eax                     ;ss
        mov     ax,OFFSET IDATA:ustack  ;sp offset to end of stack
        push    eax                     ;sp
        mov     eax,(11b SHL 12) OR (1 SHL 17) ;VM, IOPL==3
        push    eax                     ;flags word
        mov     ax,ITEXT                ;back to the real world
        push    eax                     ;CS
        mov     ax,OFFSET vmstart       ;start of user code
        push    eax                     ;IP
        clts                            ;
        sub     eax,eax                 ;flags0
        push    eax                     ;
        popfd                           ;clean out flags - NT must be 0
        iretd                           ;into virtual mode!
assume  cs:ITEXT,ss:nothing,ds:nothing,es:nothing,fs:nothing,gs:nothing
;Now we're in V86 mode. Return to DOS!
vmstart:xor     al,al                   ;unmask all interrupts
        out     21h,al                  ;
        out     0a1h,al                 ;
        sti                             ;set interrupts!
        mov     ah,31h                  ;TSR
        mov     dx,400h                 ;16k-enough?
        int     21h                     ;
failpm: mov     ax,IDATA                ;
        mov     ds,ax                   ;
        ASSUME  ds:IDATA
        mov     ah,9                    ;no - print message
        int     21h                     ;
        mov     ax,4C01h                ;failure
        int     21h                     ;exit
```

```
;adjust converts our idt/gdt tables into correct form
ASSUME  cs:ITEXT, ds:nothing
adjust: add     si,2                    ;skip limit-it's ok
        mov     di,si                   ;point at it with di as well
        lodsw                           ;get the offset
        movzx   edx,ax                  ;save it in edx
        lodsw                           ;get the flags
        mov     bx,ax                   ;save in bx
        lodsw                           ;get the segment base
        test    bh,00010000b            ;segment descriptor
        jnz     @F                      ;yes, ok to adjust
        test    bh,00000100b            ;gate
        jnz     adjust                  ;no, must be a gate- no adjustment
@@:     movzx   eax,ax                  ;extend to 32 bits
        shl     eax,4                   ;adjust for paragraphs
        add     eax,edx                 ;calculate absolute to edx
        stosw                           ;save the lin_addr_lo
        shr     eax,16                  ;get the high end to dl
        xchg    ah,bh                   ;get the access byte,
        stosw                           ;0 to lin_addr_xh
        mov     ax,bx                   ;
        stosw                           ;
        loop    adjust
        ret
ITEXT           ENDS
;*************************************
END     start
```



All code in this issue is available through our bulletin board services or directly from *PJ* on disk. For more information see pages 8 and 88.

TechnoJock's Turbo Toolkit

Attention Turbo Pascal programmers.....

Cut your program development time in half and create sophisticated user interfaces in minutes with *TechnoJock's Turbo Toolkit!*

The *Toolkit* contains twelve units with all source code included, and a 300 page manual full of examples. These range from a comprehensive unit that provides a Sidekick Plus style directory window with all the expected sorting features, to a basic procedure that writes text centered on the screen.

And it's easy to use! If you've bought other toolkits but didn't use them because of all their complicated rules ... TTT is for you.

TECHNOJOCK SOFTWARE

- **ultra fast screen writing**
- **EGA/VGA extended lines**
- **exploding boxes**
- **virtual screens**
- **sorted directory listings**
- **comprehensive form input**
- **pulldown menus**
- **nested menus**
- **string manipulation**
- **windows**
- **full mouse support**
- **keyboard control**
- **date manipulation**

"TechnoJock's Turbo Toolkit may just cut your work in half! It provides easy-to-implement procedures.... designed for all Turbo Pascal programmers. With this Toolkit everyone becomes a TechnoJock."
Turbo Users Group

"Here, despite the whimsical name, is shareware at its best: an innovative product, a superb manual, quick mail-order service, and a reasonable price... If you're programming in Turbo Pascal, you absolutely must have TTT!"
Public-Domain Software & Shareware 2nd edition

"Looks great! We have decided to implement your Toolkit in in-house programs for Tech Support."
Rich Kubota
Technical Support
Borland International

TechnoJock Software, Inc.
PO Box 820927
Houston, TX 77282-0927

Orders: 1-800-242-4775
Enquiries: (713) 493-6354

Only $49.95!

Plus $5.00 shipping and handling (Canada $10.00, overseas $15.00). Texas residents add 8% Sales Tax.

Send checks to TechnoJock Software. Send Visa/Mastercard details to: Public (Software) Library PO Box 35705 Houston TX 77232-5705

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Porting programs to run under a 386 DOS extender

Converting to 386 protected mode

By Ronald Berry

Will your program run faster in 80386 *protected mode* than it does in real (8086) mode? You'd better believe it! Is this, finally, a free lunch? Not quite, but the cost of conversion isn't all that great. Does that mean that it's not too hard for Peter Norton or Steve Baker, or are we talking about normal folks? Let's see what you, the reader, think.

This article is more a case history of a specific project than a comprehensive discussion of all of the steps necessary to convert absolutely any MS-DOS program to 80386 protected mode. Many readers can stop right here because a well-behaved application written entirely to ANSI C standards should require no changes. If your code is truly clean, it will compile and run with no trouble whatsoever.

Normal people should continue to read, though, because this discussion is based on notes compiled while converting a commercial graphics package to run in 80386 protected mode. The graphics package, like many real life sets of functions, uses mixed C and assembly language to achieve very fast, highly specialized execution code. It was apparently written in Microsoft C and MASM, and makes use of many of the tools and macros available to those packages.

OBJECTIVELY SPEAKING

While the process required for this type of conversion is not all that difficult, I will focus on some of the trouble spots. After all, what good is an article which says, "There's nothing you need to know"?

The goal of this project was to convert a graphics package to compile with MetaWare High C 386 and assemble, where necessary, with Phar Lap 386|ASM. The resulting library needed to link with Phar Lap 386|Link, and the result then had to run under Phar Lap 386|DOS-Extender (support for the Eclipse OS/386 DOS extender would be added sometime later). Embodied in these constraints was the implicit requirement that the converted code perform in exactly the same manner as its 8086/80286 predecessor had.

OUR FIRST RULE

The 80386 programming environment is still evolving. Soon, there may be a CodeView-like debugger for protected mode C (both WATCOM and MicroWay C/386 are soon to offer one). Maybe an 80386 assembler will someday have such a tool, or maybe even macros similar to MASM. It's amazing how much we come to depend on these things. That dependence is even more clearly illustrated when your original assembly language code was written to use a proprietary macro package which in turn relies on the suddenly unavailable MASM macros for final code generation. These statements should not be interpreted as complaints about software vendors. We *all* write our MS-DOS code in some way similar to this. So, rule number 1 for any 80386 protected mode conversion project is:

Analyze the original code for Microsoft or other proprietary dependencies and eliminate them before starting the conversion. Do it in real mode while you still have tools like CodeView to make the job easy.

A good C compiler like High C 386 will require no changes to ANSI standard C source code. In most cases, the compiler will issue a few warnings to point out some sloppy coding habits and deliver an object file as expected. The compiler makes the assumption, though, that you haven't deviated from the intent of the ANSI coding standards, and that you have avoided misuse of system calls such as INT86.

DATA SIZES

Idiosyncrasies in this particular project pointed out one of the fundamental differences between 8086 code and 80386 protected mode code: 80386 *ints* are 4 bytes wide. If your code

---

Ron Berry is president of Peerless Engineering Service, which just moved from Soquel, California to 541 Willamette, Suite 304, Eugene, Oregon, (503) 344-8815. Peerless consults on scientific and engineering software for mathematics, graphics, robotics, and machine control.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

is well-behaved, this will make no difference to you. The rest of us should consider the following structure:

```
struct inventory {
     int stock_num,
     int actual_count,
     int optimum_supply,
     int daily_use;
     } red_widget;
```

In real mode the "sizeof" this structure type is 8 bytes, but in protected mode it is 16. If some of your operations depend on 2-byte spacing, or some other size-dependent measurement, elusive bugs will begin to pop up. Clever time-saving techniques such as ANDing red_widget[2] with red_widget[4] no longer provide useful information because the storage boundary has changed. Sure, you can make the elements *short int* and *pack* the structure, but you have to remember to do it. Why do protected mode compilers generate 4-byte *ints*, anyway? Because that's how the 80386 CPU works best, and the *int* is defined as the most efficient data type in C, so that's the right size for it.

Since we are on the subject, let's discuss data structures in general. If our code performs tricks based on assumed data layout, conversion may introduce all sorts of problems. In the case of our graphics package, there were large structures shared between the C and assembly language sections of the code, with the C portion sometimes overlaying structure elements holding values assigned by the assembly language portion. The assembly code has no way to know that the C compiler has changed the data boundaries unless we explicitly provide that information. This type of data handling is not uncommon in many commercial graphics packages. Speed is paramount, and structure sharing drastically reduces argument handling. It is imperative that all dependencies of this type be thoroughly documented before starting your conversion.

Tracking a changing value across two screens in two languages is relatively easy under CodeView. However, the Phar Lap 386|Debug tool is only marginally better than Microsoft's long-obsolete SYMDEB, so this type of debugging soon becomes very tedious in protected mode. Intuitive leaps to error locations quickly become toad-like flops in the morass of graphics retraces on the single screen which may be addressed. So, rule number 2 for 80386 protected mode conversion is:

Isolate all code which uses offsets to data locations or bitwise operators. Then analyze each operation for data size dependency. Ameliorate those dependencies by checking for "sizeof" before performing the operation.

Of equal importance is rule 2.1 which states:

Do not take short cuts into data structures of any kind, and make sure that unions are extensively documented.

INTERRUPT CONFLICTS

There are a number of conflicts between MS-DOS BIOS calls and 80386 and 80387 hardware exceptions. These are not significant until you begin to program in 80386 protected mode. The Phar Lap 386|DOS-Extender interrupt vector table defaults to an arrangement that avoids certain known conflicts. One of the more interesting of these conflicts is INT 05h (BIOS Print Screen function). You may have used this interrupt in real mode as an argument to the INT86 function available in most C compilers to print the screen. The same call under the Phar Lap 386|DOS-Extender will generate a BOUND range error! Why has Phar Lap messed up a valuable and successful tradition? They haven't; this is the way the interrupt table should have been arranged all along. It's just that 10 years of error have finally caught up with us.

By default, the Phar Lap DOS extender corrects for the historical MS-DOS interrupt and Intel exception vector problems by relocating all conflicting vectors to table locations between INT 70h and 80h. Unfortunately, they were not moved in any predictable order that I can discern. For example, INT 05h became INT 80h, but INT 08h became INT 70h. A comprehensive list of DOS/Phar Lap interrupts is included on the code disk for your convenience. In any event, this leads us to rule number 3 for 80386 protected mode conversion:

Isolate and analyze any system calls, such as (but not limited to) INT86, while making a careful study of the DOS Extender Interrupt Table listing.

The corollary to this rule, designated rule 3.1, is:

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

10 years of error have finally caught up with us.

Isolate and analyze all inline code generation statements while making a careful study of the DOS Extender Interrupt Table listing.

CLEVER CODING AND MEMORY

Each of us will probably discover some new form of deviant coding to make C source code difficult to use with an 80386 protected mode compiler, so I can't cover them all. One point comes clearly through all of my examples, though. Programs which use C in a "legal" manner will convert very easily.

Those programs which embody fiendishly clever techniques to circumvent language safety features or to modify operating system parameters will be very difficult to convert safely. Access to our imaginary inventory structure is a clear example of this type of problem. Rather than dream up tricky ways to compare parts of elements to save time, just compare the appropriate elements! Sure, this takes some extra operations, but it works all of the time. This leads to the most onerous of my rules, rule number 4 for 80386 protected mode conversion:

If a particular module or section of code proves troublesome to convert, that is a sure indication that the underlying logic is faulty. Rethink and rewrite the entire section. Continued attempts to change "just one little thing" will almost never prove to be efficient.

With reference to the C portion of our conversion, one last change in thinking must be mentioned. It is of profound significance:

Forget all about memory models!

In 386 C compilers, near pointers are four bytes long. This means that you can address a contiguous memory space of four gigabytes without resorting to segmentation. HIGH C 386 and WATCOM C/386 offer other memory models, but they all carry the execution time penalty of segment addresses. Four bytes is the optimum pointer size for an 80386, and it provides access to more memory than most of us can afford this week. I see no compelling reason to use larger pointers.

CONVERTING ASSEMBLY CODE

That's about all I found out about converting C programs to protected mode. The assembly language portion of the job was a bit more intricate.

Data Junction® converts data files!

Any ASCII • dBASE • R:BASE • Lotus 1-2-3
Excel • Paradox • Informix • Q&A • Oracle • Xdb
askSam • SC4 • Clarion • DIF • SYLK • Symphony
ACT! • Platinum • DAC • and other application files
IBM/Mips/Sparc/Cray/Vax/Intel/68000 Binaries
Btrieve • c-tree • EBCDIC • Packed data
. . . and more!

- *No programming-completely menu driven!*
- *Select and rearrange records, fields and characters!*
- *Instant script file creation for total batch processing!*
- *Map new data structures automatically by name!*
- *Ideal for OEM applications!*

". . . an invaluable integration tool." -Byte
". . . more than worth its price." -PC Magazine
". . . a surefire winner." -Computer Language

Standard $99 - dBASE, ASCII, 1-2-3, Q&A, Enable, DIF, askSam & merge files for WordPerfect, Word & WordStar

Professional $199 - Standard formats plus Binary/EBCDIC, SC4, SYLK Paradox, R:BASE, Clarion, Excel, c-tree & Btreve

Advanced $299 - Standard & Professional formats plus Informix, Xdb, Oracle & C-ISAM

Call **800/444-1945** or 512/482-0824
MC/VISA • AMEX • COD • Corp. PO.
available for LANs, Xenix & Unix
Tools & Techniques Inc. • 1620 W 12 • Austin TX 78703

T&T TOOLS & TECHNIQUES

DON'T RISK IT!

If you're a software developer, you are risking more than you think if your product has a primitive installation system.

In today's competitive market, your product **must** be easy to use -- **immediately**. It is no longer commercially acceptable to tell the end-user to use batch files, COPY, DISKCOPY, RESTORE, or to require wading through paragraphs of text in the manual. The installation process is something that a program developer **should** and now **can easily** completely automate.

NEW VERSION 3.0!

OS/2 and International versions now available!

INSTALL 3.0 is a bullet-proof automatic installation program you can distribute - royalty free - with your product. Features: Up to 4.3 gigabyte file sizes • Windowing/color user interface • **Includes C source, but no programmming is normally needed** • File integrity verification • Intelligent handling of virtually all errors (open disk drive doors, full disks, unformatted disks, etc) • Hard & floppy disk installation • Multiple source & target disks • Multiple target subdirectories (which *INSTALL* will create if necessary) • Unlimited free tech support • Granular installation • Easy method of adding new functionality • Ability to execute DOS commands • Can intelligently modify CONFIG.SYS & AUTOEXEC.BAT files • **Data compression** • Automatically detects the CPU, DOS version, monitor, RAM, ANSI.SYS, LIM, etc • **30-Day Money-Back Guarantee.**

INSTALL offers 4 years of proven reliability that was used to install over millions of copies of products in 1989 including some of the most prestigious in the industry. We would like to add your name to that list.

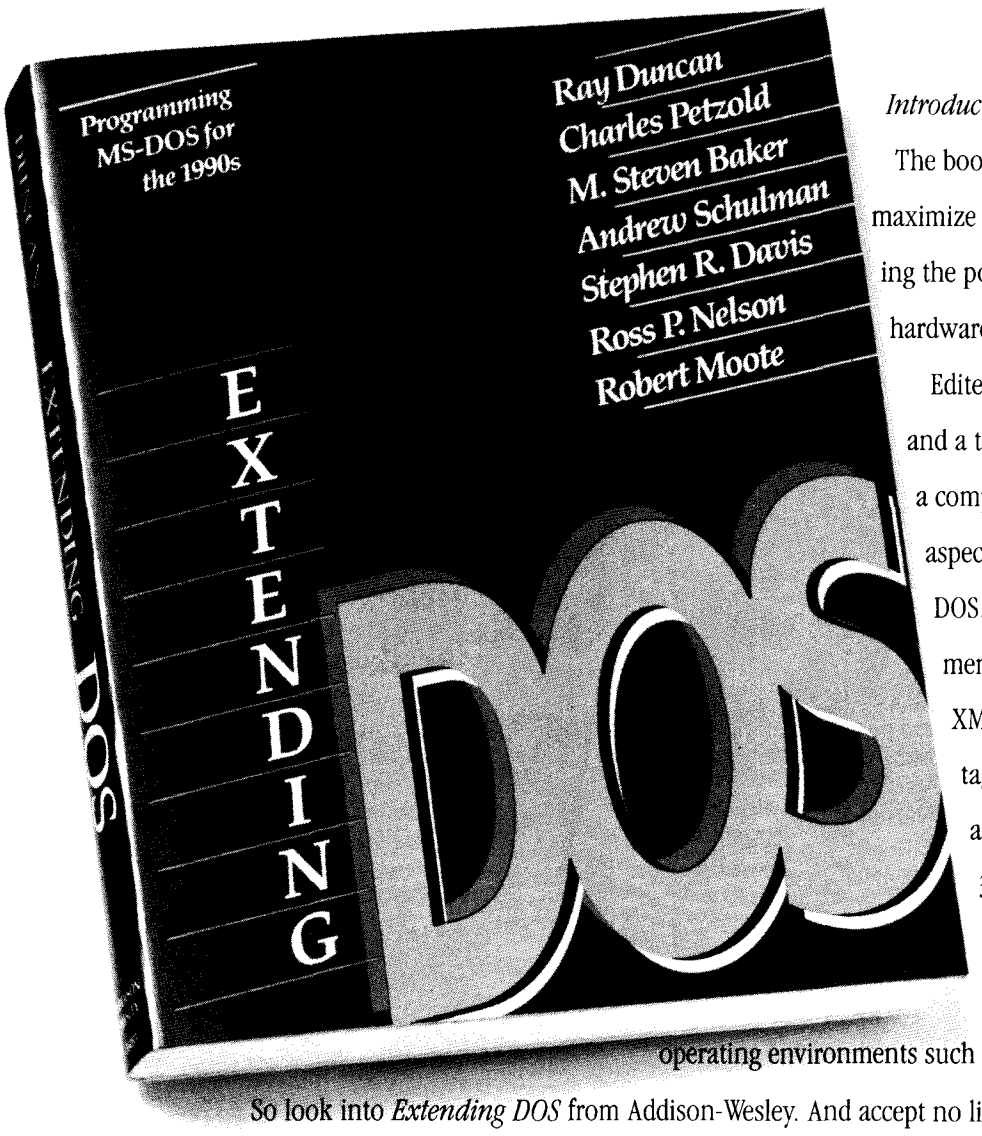
ORDER NOW TOLL FREE 1-800-331-2783

KNOWLEDGE DYNAMICS CORPORATION
Highway Contract 4 Box 185-H, Canyon Lake, TX 78133-3508
International: 1-512-964-3994 24-hr FAX: 1-512-964-3958
MasterCard/VISA/COD/Purchase Order

$249.95

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Accept No Limitations.



Introducing Extending DOS.

The book that enables programmers to maximize program performance by harnessing the power of a new generation of PC hardware to DOS applications.

Edited and coauthored by Ray Duncan, and a team of experts, *Extending DOS* is a complete and authoritative guide to all aspects of extending and enhancing DOS. It will show you how to address memory above 640K with EMS and XMS standards, how to take advantage of the speed and increased addressable memory of 286 and 386 architecture, understand industry standards such as VCPI and DPMI, and program for operating environments such as Windows and DESQview.

So look into *Extending DOS* from Addison-Wesley. And accept no limitations.

Available wherever computer books are sold. 448 pages, paperback; suggested retail price, $22.95; ISBN 0-201-55053-9.

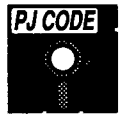
ADDISON-WESLEY

Addison-Wesley Publishing Company, Inc., Route 128, Reading, MA 01867.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
; Sample code for Hercules display segment creation using
;       386|ASM assembler
;       Designed to run under the Phar Lap 386|DOS-Extender

;------------------- constants -----------------------------------
INDEX       EQU     03B4H                   ; Port Addresses
CNTRL       EQU     03B8H
LOCK        EQU     03BFH

SCREEN_ON   EQU     8                       ; Control Codes
GRAPHIC_MODE EQU    3
TEXT_MODE   EQU     21H


_data  SEGMENT WORD PUBLIC 'DATA'
GSEL        DW      01CH                    ;Phar Lap & Ergo segment
                                            ; selector for screen buffer
                                            ; not constant in case
                                            ; set_herc needs to change it

; Hercules control codes
gtable      DB     35H, 2DH, 2EH, 07H, 5BH, 02, 57H, 57H, 02, 03, 0, 0

_data ENDS


_text SEGMENT WORD 'CODE' PUBLIC

_set_herc_mode PROC NEAR
            push    esi

            mov     edx, LOCK               ;unlock current mode
            mov     eax, GRAPHIC_MODE
            out      dx, al                 ;unlock graphics mode
                                            ; and include page 1
            lea     esi, gtable             ;for 2 page ops
            mov     ecx, 2000H
            xor     ebx, ebx
            call    _set_hercules

            pop     esi
            ret
_set_herc_mode ENDP

_set_hercules  PROC NEAR

            push  ds
            push  es
            push  edi
            cld

            push  eax
            push  ebx
            push  ecx
            push  eax               ;save the Graphics mode twice

finder:                             ;create a segment for card
            mov   eax, 4800H        ;first create a 0 length segment
            mov   ebx, 0
            INT   21H               ; selector returns in AX
            mov   GSEL,ax           ;save it in our segment selector

            mov   es,  ax           ;give it address and size
            mov   ebx, 0B0000H      ; address (Herc base)
            mov   ecx, 10H          ; size = 64K (16 pages)
            mov   eax, 250AH        ;386|DOS-Extender extended INT21
            INT   21H               ;GSEL now holds a segment selector
                                    ; number of unknown value which
                                    ; points to a 64K segment with an
                                    ; absolute base address of B0000H

            pop   eax               ;finish the card initialization

            mov   dx, CNTRL         ;turn off the screen
            out   dx, al            ;initialize the 6845
            mov   dx, INDEX
            mov   ecx, 12           ;use 12 values from gtable
            xor   ah, ah            ;send 0 first
parms:      mov   al, ah            ; and now send gtable
            out   dx, al
            inc   dx
            lodsb
            out   dx, al
            inc   ah
            dec   dx
            loop  parms

            mov   ax, GSEL          ;screen base to ES
            mov   es, ax            ;this is our segment selector
            xor   edi, edi
            pop   ecx               ;8000 bytes to the screen
            pop   eax               ;pop EBX value to fill screen
            rep   stosd             ;doing 2000 x 4 bytes

sback:      mov   edx, CNTRL        ;screen back on
            pop   eax
            add   al, SCREEN_ON
            out   dx, al
            pop   edi
            pop   es
            pop   ds
            ret
_set_hercules  ENDP

_text ENDS
```



All code in this issue is available through our bulletin board services or directly from *PJ* on disk. For more information see pages 8 and 88.

The first thought that you will have as you look at 100,000 bytes of virtually uncommented assembly code is, "Can I think of a way to mechanize this process?" The answer is, "Yes and no." Hindsight convinces me that Phar Lap is right in suggesting that conversion time is a good time to completely rewrite assembly code for protected mode applications. A complete rewrite may not fit your budget or your calendar, but it should at least be considered.

Mechanizing the conversion of existing code may appear to save time, but the resulting code will be far less efficient than if it were properly rewritten for this new environment. Mechanized conversion will quickly devolve into an immense debugging project for which only primitive debugging tools are available.

SEGMENT REGISTER USE

Let's look at a few of the major considerations in this part of a conversion to 80386 protected mode. We'll start by examining the proper and most efficient use of the segment registers.

Most of us learned to use 8086 segment registers not only for reaching additional memory, but also as an organizational tool, as base pointers for data access. Sometimes segment registers are used to reach areas of memory allocated for system functions, or memory locations allocated to add-in boards. These same uses are available in protected mode, but the initial methodology for defining the segment address is very different. Adjusting to a new methodology always requires some expenditure of time and effort. Much of this adjustment cost can be avoided by eliminating the requirement for the change. As an example, you can immediately change your thinking on data access. All addresses are now NEAR, so you can eliminate all use of segment registers for data access.

Segment registers are also used for other purposes, however, so the algorithm for deciding whether to remove them

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

mechanically has to be a bit more than "globally change all 'ES:' to blanks." Each and every use of a segment register must be separately analyzed. With that in mind, let's look more carefully at the single most significant difference between 8086 and 80386 protected mode programming: the use of segment registers.

ELIMINATE SEGMENT BASE ADDRESSES

Any memory location within 4 gigabytes can be reached with a NEAR pointer, so all of the convolutions formerly used to establish segment base addresses are unnecessary and should be eliminated. This blanket recommendation may seem excessive in the face of a clear capability to establish multiple data segments. If our only use of segmentation was to access actual defined segments, we might elect to suffer the loss of time required to use far pointers, but many 8086 programmers assign segment register values just by placing a label address in a segment register and then offsetting for access. Unfortunately, protected mode segment registers can't be set that way.

As you know, 80386 protected mode uses segment descriptor tables to hold the definition of each active segment. The selector number placed in a segment register is just a pointer to the Local Descriptor Table entry for that segment. Selector numbers are usually two-byte values, so if we casually place a label address such as 14396h in there, all we are going to see at run time is the dreaded "Memory Protection Error" message as the machine locks up. This problem is easy to correct, but each use of a segment register should be carefully inspected.

Both Phar Lap and Eclipse DOS extenders have made special provision for access to video memory, where segmentation *is* a proper method, but that is a special and well-documented case. This leads to rule number A1 for 80386 protected mode conversion:

Attempt to eliminate all use of segment registers for data access in protected-mode-only programs.

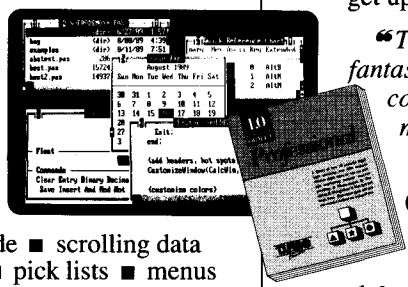
A similar illustration of the case against mechanized conversion would be a global change of all references to SI to ESI. This change is ultimately necessary, but a blanket change of all cases of **mov ax,[si]** to **mov ax,[esi]** attempts to **move** four bytes rather than two, and you may not want that for some reason. What if we changed all **ax** to **eax** at the same time? Might work, but you still need to look at each change.

I found it mentally convenient to include TYPE PTR operators where I might otherwise not have used them, just as an aid to clarifying the original intent of the code. So, as you see, all changes will have to be analyzed anyway. Why not optimize them at the same time?

Save "Man-Years of Effort" with Turbo 5.5

Don't Start from Scratch with Object-Oriented Pascal

Object Professional is a huge library of over 200 object types and 2000 methods that will multiply your productivity. Window object types let you use overlapping and resizeable windows. The windows include ■ scrolling data entry screens ■ pick lists ■ menus ■ file selection ■ printed forms ■ help capability and more.



Build your programs using proven data object types like stacks, linked lists, virtual arrays, and more. System-oriented routines provide swappable TSRs in only 6K of RAM, EMS management, and much more.

Object Professional includes clear, comprehensive documentation, on-line help, full source code, technical support, and hot demo programs. Pay NO royalties. You'll get up to speed fast with OOP!

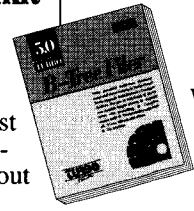
"The range of objects is fantastic. Object Professional could literally save you man-years of effort."

Jeff Duntemann

Object Professional 1.0, only $150.

A Multi-User B-Tree Toolkit

Write powerful network compatible databases faster and easier using **B-Tree Filer 5.0**. You'll have the fastest, safest, most flexible databases – no rigid structure, no TSR hassles, no running out of files. And they're compatible with Novell, 3Com, MS-NET, and others.



You get ■ Fixed and variable length records ■ Two billion records per database ■ Up to 100 indexes per index file ■ Fail-safe mode with journaling ■ Units for sorting, browsing, reindexing, and network control.

B-Tree Filer includes full source code, documentation, technical support, and you pay NO royalties.

"B-Tree Filer... a well rounded, feature-rich approach to B-Tree databases."

Computer Language, 1/90

B-Tree Filer 5.0, only $125. (single user)
With network support, $175.

Call toll-free to order.
1-800-333-4160

Satisfaction guaranteed or your money back within 30 days. Add $5 per order for shipping in U.S. and Canada. Inquire about other shipping charges. OPro requires Turbo 5.5. BTF requires Turbo 4.0, 5.0, 5.5, or QuickPascal.

TURBO Power

8AM - 5PM PST Monday through Friday, USA & Canada.
For more information call (408) 438-8608. Fax: (408) 438-8610.
TurboPower Software PO Box 66747 Scotts Valley, CA 95067-0747

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

My recommended list of changes to assembly code for the next Real to Protected mode conversion job looks like this:

- Eliminate all unnecessary data segmentation.
- Eliminate all unnecessary use of segment selectors.
- Remove all DS initialization code no longer necessary.
- Widen all possible register references to Extended Form. This results in more efficient code generation.
- Widen all jump tables to double words to accommodate the 4-byte pointers which the elements represent.
- Locate all references to jump tables and widen the jump target references to four bytes (see the code fragment below).
- Inspect all arithmetic operations for changed result locations caused by register extension.
- Inspect all BIOS calls for corrected interrupt vector locations (check the DOS extender interrupt vector table).
- Modify all video access calls to use the Phar Lap or Eclipse video segment selector. If a non-BIOS adapter such as a Hercules monochrome adapter must be accommodated, be sure to properly create a segment to describe it. Sample code is given in Listing 1.
- Modify other special segment uses for proper selector definition.

The fragmentary example below illustrates some of these considerations:

Typical REAL mode code

```
_data SEGMENT WORD PUBLIC 'DATA'
res_jmp_table  DW     offset MRES
               DW     offset HRES
               DW     offset ERES
_data ENDS
      .
      .
               mov    bx,2  ; bx usually comes from someplace else
               jmp    WORD PTR put_pel_jmp_table [bx*2]
```

Typical PROTECTED mode code

```
_data SEGMENT WORD PUBLIC 'DATA
res_jmp_table  DD     offset MRES
               DD     offset HRES
               DD     offset ERES
_data  ENDS
      .
      .
               mov     ebx, 2
               jmp    DWORD PTR put_pel_jmp_table [bx*4]
```

IN CONCLUSION

Many other small changes will be required by your particular application, but the items listed above seem to encompass 90% of all of the changes to be made. No, it's not as easy as adding E before some register names, but optimizing your code and doing it right is worth it. Good luck! PJ

Computer Professional's Legal Toolkit

PROBLEM:
Almost 100% of warranty disclaimers will not stand up in court!
CONSEQUENCES:
Almost Unlimited Personal Liability For User Losses!
SOLUTION:
THIS PROGRAM!

Only $159.95
Order Now!
Call 24 hours
Most cards accepted!
1-800-648-1300

R. Fringe Publishers, Inc.
Box 796, Casselberry FL 32707

FOR PROGRAMMERS, VARS, CONSULTANTS, RETAILERS, DEVELOPERS-HARDWARE/PERIPHERAL MFGS - ANY COMPUTER ENTREPRENEURS

Full Annotated Text of Laws, Court Cases and Government Regs.

Mini-Seminars

Legal Document Generator using A.I. with more than 40 forms that are valid in all 50 states

U.S. Copyrights
International Copyrights
Uniform Trade Secret Law
Patent Law
Trademark Law
Computer Crime Laws, Federal
Computer Crime Laws, all 50 States
Warranty Disclaimers
Military Computer Acquisition Regs.
Federal Computer Acquisition Regs.
Uniform Commercial Code
Legal Liability of Computer Professionals
Beta Testing Law
BBS Laws
Employer/Employee Relations
Computer and Software Leasing
Credit and Collections
Contract Bidding
Semiconductor (Mask) Work Protection
Arbitration
I.R.S. Issues Relating to Computers
Patenting vs. Copyrighting

Written by a computer-literate attorney
PRIVATE BBS • COMPUTER EMPLOYMENT EXCHANGE • UPDATES
Minimum System Requirements: MS-DOS XT, AT, 386 Compatable, with 512k Ram, Hard Drive
© 1990 Kraft & Byron

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

By Brett Glass

The world is flat

What's new in 32-bit OS/2 memory models and more

Ever since the first release of IBM and Microsoft's OS/2, developers have been pleading with Microsoft for a version that makes full use of the 32-bit capabilities of the 80386 and 80486 microprocessors. Fortunately, their prayers may soon be answered. OS/2 2.0 is now in beta and, while it probably won't be ready for prime time for another year or so, hundreds of unsuspecting developers are now serving as guinea pigs—I mean, working with it—even as you read this.

For those who aren't sure they want to shell out $2600 for the OS/2 2.0 Software Development Kit, I'll describe some of the features that will be available to those who dare venture into the brave, new world of OS/2 2.0.

NO MORE SEGMENTS

The most radical difference between 32-bit OS/2 applications and their 16-bit predecessors is the way they see memory. Segments, call gates, and other protection mechanisms unique to the Intel 80X86 family are absent; each application "lives" in a flat, 32-bit address space in which CS (the Code Segment) is the same as SS, DS, and ES (the Stack, Data, and Extra Segment, respectively). All programs use the classic "small" memory model, and the 80386's paging unit is used to control access to memory.

Figure 1 shows how virtual addresses—that is, the addresses a 32-bit process deals with—are translated first to linear, then to physical addresses. The process uses only one segment in the segmented "virtual" address space for code and data (though FS and GS may be used for special purposes, such as to address a Weitek numeric coprocessor). The program's single segment maps directly to a chunk of "linear" address space which, in turn, is composed of 4K *pages*. Each page may lie anywhere in physical RAM, or may be swapped out to disk if virtual memory ("storage overcommitment," in OS/2 parlance) is being used.

How are shared code, shared memory, DLLs, and protection implemented under this scheme? Figure 2 gives an overview of the address spaces of several 32-bit processes, as they're set up by the 80386 paging unit. Each process has a "private" region which contains its executable code, private data, and stacks. (The pages containing the executable code for an application may be mapped into the address spaces of several processes, as can shared memory.) The code for DLLs is mapped into the address spaces of all the processes. The "instance data" for the DLL (that is, the data which the DLL allocates on a per-process basis) isn't shared among processes, but resides at the same address in every process. This is handy because addresses embedded within the DLL code can be fixed up once, when the DLL is loaded, and will point to the right place in each client's address space.

PROS AND CONS

The OS/2 2.0 memory model has several advantages in terms of speed. It's no longer necessary to load segment registers—a time-consuming process due to the many security checks enforced by the processor. "Near" jumps and calls—which can take up two bytes less than their "far" counterparts—may be used to call routines within a process and also to call DLL routines. Arrays larger than 64K are easy to access because there's no need to calculate a segment selector for each 64K "chunk" of the array. Stack sizes for individual threads don't need to be calculated in advance: the OS can detect when the stack pointer enters the lowest page of the stack (by setting up a "guard page") and allocate more room.

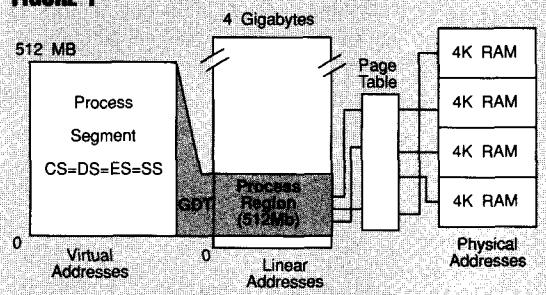
The OS also uses virtual memory more efficiently with paging. On the 80286, the contents of memory must be swapped to and from disk as entire segments of up to 64K in size. Swapping 4K pages instead lets the system read and write smaller chunks, and lets the OS use more highly optimized disk routines to do it. Finally, if OS/2 is ever ported to other microprocessors, the flat memory model and 4K pages (the same size used on most other MMUs) should make the job easier.

On the minus side, this new model throws away some of the best protection features of the Intel architecture, mechanisms which often catch buggy code before it does serious damage. With segments, an illegal memory access even one byte outside the segment's limits is trapped before it causes harm. If you put an array in its own segment, you can detect attempts to access memory beyond either end of the array. With paging, on the other hand, the system can only restrict access to whole pages of memory. Thus, if your array is 3K long and is positioned at the

Brett Glass is a free-lance writer, programmer, and consultant residing in Palo Alto, California. He can be reached at rogue@well.sf.ca.us on Internet, on CompuServe at 72267,3673, or as glass on Bix.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 1



Creation of a "flat" 32-bit address space for OS/2 2.0. A process uses only one segment, which is addressed by CS, DS, ES, and SS. It's the page table which determines what parts of the physical memory the process can "see," as well as how each page may be accessed.

Your operating system finally caught up with our editor.

Introducing BRIEF for OS/2. It's the most versatile editor available, in a powerful new environment.

Now, you can have the best multi-level undo available, potent search and translate capabilities, and two powerful (but easy-to-use) macro languages. Plus all the inherent advantages of OS/2, like multitasking.

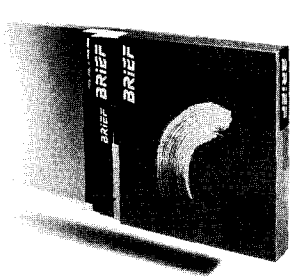
And BRIEF is still easy to use, and easy to learn. Commands are consistent and intuitive, and the documentation is clear and concise. Best of all, it's still affordable—just $195.

If you've made the investment in time and money to move up to OS/2, make sure you're working with an editor as advanced as your operating system.

Call today to order your copy of BRIEF for OS/2. And stop playing catch-up.

Special discount for DOS BRIEF users.

Call 1-800-343-7308.



BRIEF® for OS/2

UnderWare, Inc., 321 Columbus Ave., Boston, MA 02116 (617) 267-9743

beginning of a page, your program can overshoot by up to 1K before the error is detected. What's more, even if the program does "run off the end" of a page when it shouldn't, the error will only be caught if the next page is not being used.

Many programmers are more than happy to accept these tradeoffs, having been convinced that segments are bad by the problems of the 80286 (which had more to do with limits on segment size than with segments per se). But those whose subtle bugs have been caught by earlier versions of OS/2, or who want the highest possible degree of protection, may wish to stick with the 16-bit OS/2 programming model (which will also be supported in OS/2 2.0) for their most critical code.

OTHER GOODIES

The 32-bit API is mostly an extension of the 16-bit one, with references to segments removed and some new calls involving paging added. But some facilities, such as semaphores, are significantly different. The new API adds three new kinds of semaphores: *Mutex*, or mutual exclusion semaphores, which prevent two threads or processes from gaining access to a critical resource at the same time; *Event* semaphores, which simply signal that an event has occurred; and *Mux* semaphores, which allow a thread to wait on several other semaphores all at once (doing this was especially awkward with OS/2 1.2's initial implementation of semaphores). While the 32-bit semaphore functions are by and large much more useful than the 16-bit versions, they're also completely incompatible with them. Therefore, applications using semaphores must be changed when they're ported to 32-bit mode.

OS/2 2.0 provides language-independent support for exception handling. This mechanism makes it easy to trap exceptions generated by certain keystrokes (e.g., Ctrl-Break and Ctrl-C) and errors (such as protection faults and divide overflows). It also handles C's **setjmp**() and **longjmp**(), Pascal's non-local gotos, and Ada's exceptions without the need for large amounts of support code in the application.

Another piece of good news is that the maximum number of threads in the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Tired of the limitations of MS-DOS?
Thinking about switching to another OS?
Well DON'T, now there is:

PRO-CLI v4.1

CLI is a tiny, intelligent front & back end processor that enhances the MS-DOS system kernel. CLI provides many features of advanced OS's, like VMS & UNIX, to MS-DOS without the tremendous costs and additional training.

CLI-Types: ALIASs, LOGICALs, SYMBOLs, LEXICAL-FUNCTIONs and STREAMs are available to the command line, batch programs, & all existing application programs. CLI is 100% MS-DOS compatible, and NO changes are required in any software to reference any CLI-Types.

Some of the features offered by PRO-CLI include:

- batch programs with numeric & string processing
- facilities for source code and project management
- transparent task-to-task communications
- file & device management from the command line
- command line editing, history, and recall
- flexible system installations & re-configurations
- user configurable PRO-CLI resources

For the first time, designers, developers, and managers can coordinate software development & production efforts without the major restrictions of MS-DOS.

Call or Write for Additional Information!

Trademarks: VMS of D.E.C. MS-DOS of Microsoft UNIX of AT&T

Alphax Systems, Inc.
1005 Sussex Blvd.
Broomall, PA 19008
(215) 543-2658

Price: $ 149
S/H: $ 6

Major Credit Cards Accepted

Circle **345** on Reader Service Card

Multitasking

Do the multitasking systems you are considering using seem a bit like overkill or need an awful lot of memory? Maybe all you really need is a system which can run more than one DOS program concurrently. If this sounds like your situation, consider MultiDos Plus. Flexible preemptive scheduling of multiple tasks with full access to MS-DOS or PC-DOS. LIM 4.0 support. For more information please call for a free brochure.

Nanosoft
800-678-2141
508-651-0091
FAX 508-655-8860
13 Westfield Road
Natick MA 01760

Circle **109** on Reader Service Card

GRAPHICS!

INTRODUCING THE GX DEVELOPMENT SERIES

Hand coded in Assembly Language, these toolkits are designed to work alone or together with no code overhead. They support 8 languages and over 30 resolutions up to SuperVGA, for the fastest, tightest code. Save yourself time and money with our excellent tools and support.

An Exceptional Product Prog. Journal. 7/89

Powerful Image Graphics DBAdvisor. 1/89

Intuitive and Useful BYTE. 9/89

Well worth the investment Dr. Dobb's. 8/89

Excellent set of Libraries Comp. Language. 9/89

New Product! **GX Graphics** **$149**

A complete graphics library supporting all graphics primitives from *grPutPixel* to *grDrawEllipse*. Full support for logical operations (including rubber-banding), clipping, and drawing to off-screen virtual buffers in conventional memory or expanded (LIM 4.0). The cornerstone of the GX Series, use this graphics library instead of the BGI or MS libraries and make your program faster, smaller, and more portable across compilers–while supporting more modes.

New Version! **PCX Programmer's Toolkit 4.0** **$249**

The powerful and popular PCX toolkit is now even better! Over a dozen new functions have been added, more display adapters and compilers, and optimized assembly language for more speed and less code. Now a part of the GX Series, new features include image sizing and scaling, access to low level encode and decode, enhanced virtual buffer support, dot matrix /laserjet printing of scanned images, more examples (including real-time animation), plus all of the previous features that make the toolkit so popular. If you need PCX support, you need this.

Effects! **PCX Effects** **$99**

Add Special Effects to your PCX Toolkit Applications. Fades, wipes, animation, and music.

Fonts! **PCX Text** **$149**

Blazing graphics text in 80 different fonts, with bolding, underlining, etc.

All packages support 12 compilers for C, Pascal, Basic, Fortran, Assembly, Modula-2, Prolog, and Clipper. All modes of the Hercules, CGA, EGA, MCGA, VGA, and Super VGA adapters are supported, up through 800x600x256 (30+ modes in all). Assembly Language source code is optionally available. Trademarks are property of their respective holders.

Order Your Copy Today!

1-800-227-0918

NO ROYALTIES! **30-DAY GUARANTEE!** **SOURCE AVAILABLE!**

COD CHK PO

GAMS MICROPROGRAMMING

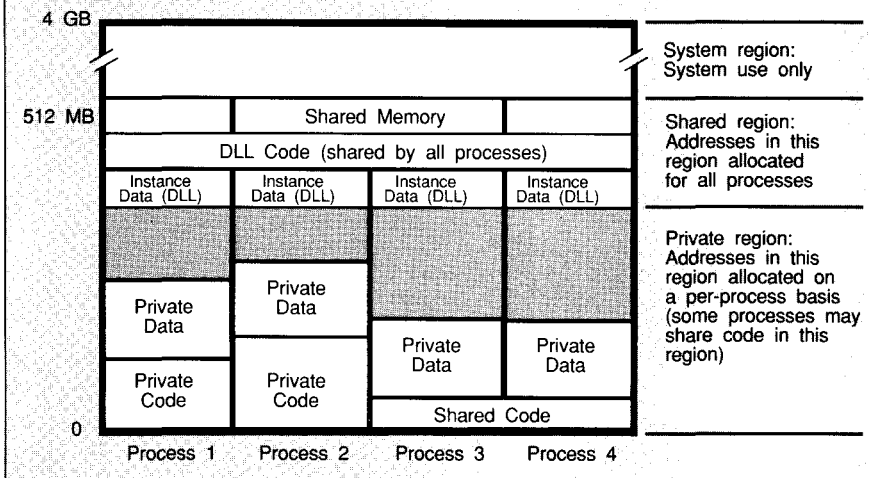
11315 Meadow Lake
Houston, TX 77077
(713) 870-0737

FAX: (713) 870-0288
BBS: (713) 266-9362

Circle **261** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 2



The "flat" address space seen by 32-bit OS/2 processes. There are three regions: the Private Region, in which addresses are allocated on a per-process basis; the Shared Region, where ranges of addresses are allocated on a system-wide basis; and the System Region, which is only accessible to code running in ring 0.

system has been increased to 4K, with no limit on the number of threads per process. Those developers who avoided taking full advantage of multi-tasking for fear of setting off too many concurrent threads can now relax.

Finally, OS/2 2.0 will provide 80387 emulation automatically if the system doesn't have a math chip installed. Thus, your OS/2 2.0 programs won't ever need to include their own floating point routines.

THE "WINDOWS BOX"

At a press conference on April 30th, Microsoft's Steve Ballmer, Senior Vice President in charge of Systems Software, confirmed the existence of the much-rumored "Windows Box." This is a feature which has the potential to let OS/2 2.0 run protected-mode Windows binaries *unchanged* within a PM window. It's in the early stages of development now, and at this writing Microsoft hasn't committed to a definite release date.

However, Ballmer has stated that this feature will *only* be included with 32-bit versions of OS/2. Developers targeting 16-bit OS/2 systems as well as Windows will still need to either port their code or use Microsoft's Windows migration tools to make applications that can run under all three environments.

BEGIN EXPERIMENTING

Whether or not you run out and buy the OS/2 2.0 SDK will almost certainly depend on what you want to achieve with your particular application. Even in 32-bit mode, a program written for OS/2 will be more sluggish than a hot-rodded DOS extender app, especially if you can get away with using a character-mapped display. And if you don't want to cut 8088 and 80286 users out of the picture, you may be forced to stick with OS/2 1.2, Windows, DESQview, or even P.O.D. (Plain Old DOS). Still, it is increasingly a 32-bit world out there. So whether or not you'll be implementing 80386 applications today, it might just be a good idea to begin experimenting with OS/2 2.0 and other environments that use the 80386's 32-bit protected mode. PJ

A professional workstation environment for OS/2®

Hamilton C shell™

"...much more powerful than CMD.EXE...blindingly fast... we have a winner...a much-needed and well-done product."
– Personal Workstation Magazine

The superior alternative to the standard OS/2 command processor. Faithfully recreates the entire C shell language as described in the Berkeley 4.3 *UNIX® Programmer's Manual*. Created explicitly for OS/2. Not one line ported from or created on anything but OS/2. Extensively multithreaded.

Features: Command line editing • History • Filename and command completion • Arrow and function keys • Enormous 64KByte command lines • Aliases and shell procedures • `PATH` hashing • Recursive filename wildcarding • Fully nestable control structures • Powerful expression grammar • Command substitution • Background threads and processes.

Over 113 commands: `alias, cat, chmod, cls, cp, cut, diff, dirs, du, eval, fgrep, grep, hashstat, head, history, label, ls, kill, markexe, more, mv, popd, ps, pushd, pwd, rm, sed, sleep, split, strings, tabs, tail, tar, tee, time, touch, tr, uniq, vol, wait, wc` and others.

Supports HPFS and long filenames.

Requires OS/2 1.1 or later. All executables will run properly in a Presentation Manager window. Not copy-protected.

$350.00. Unconditional satisfaction guarantee. MasterCard & Visa accepted. ($365.00 Canada/Mexico; $395.00 elsewhere.)

Hamilton Laboratories
13 Old Farm Road, Wayland, MA 01778–3117, U.S.A.
Phone 508-358-5715 • FAX 508-358-1113 • BIX hamilton

Circle **334** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

By M. Steven Baker

FORTRAN on the 386

A look at some DOS extender development tools

New hardware continues to outdistance software developments, so 80386 computers are often used merely as fast XT machines. The C1 step of the 80486 is now available, heralding an onslaught of 486 machines. While 16-bit versions of OS/2 are finally becoming stable, the Intel 80586 lies waiting in the wings. We find that readily available hardware is almost two chip evolutions ahead and still gaining.

This article is the first in a series on 80386 development tools. In this piece, I will focus on FORTRAN on the 386 using DOS extenders (see *Impressions*, a look at 386 DOS extenders on page 91). Future installments will focus on C for the 80386, 486 compilers, and compare tools under both extended DOS and Unix/Xenix environments.

Beginning life in 1987, DOS extenders grew out of a need to take advantage of the more powerful features of 80286 and 80386 computers without foregoing MS-DOS. Though DOS extenders are available for the 80286, my remarks will target the 386.

386 DOS extenders allow large applications to run under DOS until a true 386 operating system displaces DOS. These 386 DOS extenders set up a bridge to the DOS environment and put the 80386 chip into the protected mode so that both the larger memory space and the full 386 instruction set are available. An application built with a DOS extender is likely to be faster and have more features and power than a comparable DOS program.

FORTRAN IN THE FAST LANE

Six FORTRAN compilers support 386 DOS extenders (see Table 1). All are ANSI FORTRAN-77-compatible and provide 80387 in-line code generation for fast number crunching. Four also support the Weitek math chip. The problem with much existing source code is that it uses many early FORTRAN IV anachronisms or language extensions that are available on the forgiving IBM and VAX mainframes. Various VAX, IBM, and Unix extensions are supported by each compiler vendor. As work progresses on FORTRAN 90 (formerly FORTRAN 8X), some of these language extensions are also being supported. FORTRAN, with all of its shortcomings, continues to be the language of choice for numerics-intensive applications.

LAHEY F77L-EM/32

Lahey Computer Systems offers a full range of DOS FORTRAN compilers. F77L-EM/32, the true 32-bit compiler, generates 80386 code to run under Ergo's OS/386. All Lahey compilers support the same ANSI FORTRAN-77 syntax along with some IBM and VAX extensions. F77L-EM/32 generates in-line 80287, 80387 and Weitek numeric instructions. F77L-EM/32 is bundled with an editor, an excellent MAKE utility (Opus Make), a symbolic debugger, linker, librarian, and other utilities developed by Lahey. The compiler also comes bundled with OS/386 with the virtual memory (VM) option. For an additional $395, users are entitled to redistribute unlimited copies of OS/386 with VM along with their applications.

F77L-EM/32 features a variety of options, taken from a configuration file or overridden from the command line. This compiler reads a FIX file at startup with brief text commands, and self-patches itself at runtime. This allows a user to get a short bug fix for a problem via the phone or Lahey's bulletin board and quickly be up and running again.

The compiler generates object files based on Microsoft MASM 5.1 extensions to the Intel OMF-86 format. Lahey supplies their own 386 linker (L32) and a library manager (LM) for manipulating 32-bit libraries. The executable files created will run under ERGO's OS/386 DOS extender.

Supplemental routines are provided to handle numeric exceptions and to illustrate how to use the ERGO RPC mechanism to link with real mode code. An interface to GSS and GKS is

---

M. Steven Baker is the editor of Programmer's Journal *and a co-author of* Extending DOS, *published by Addison Wesley. He can be reached occasionally at the* PJ *offices with red pencil in hand.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Here's what they say about Zortech C++

" Zortech is a truly fine compiler...If you've been waiting for a major player to offer a professional C++ development system for OS/2 and Windows, as well as DOS, wait no longer... Zortech has it! "

Richard Hale Shaw, PC Magazine, p.38, March 13, 1990

"' Zortech has done a commendable job with C++ 2.0 and I recommend it highly...The debugger is impressive...Get the Developers version...it's worth the money."

Bruce Eckel, Micro Cornucopia, pp. 8-17, March 1990

" Zortech C++ is one of the best MS-DOS products I've had the luck to use.....I can highly recommend the Zortech 2.0 release."

Scott Robert Ladd, Dr. Dobbs Journal, pp. 64-73, January 1990

" The Zortech compiler is a well-crafted product; in particular it is efficiently documented and it produces high quality object code. I can recommend it as a fine package"

Paul G Smith, .EXE Magazine, pp. 16-22, March 1990

AT&T™C++ V2 Specification
- ✓ Multiple Inheritance
- ✓ Type Safe Linkage
- ✓ Pointers to Members

Compiler Features
- ✓ Native code compiler with separate global optimzer
- ✓ Improved MSC Source Level Compatibility
- ✓ MS Windows™ Compatible
- ✓ CodeView™ Compatible
- ✓ Fast Graphics Library with C++ interface
- ✓ Easy to use TSR functions
- ✓ Standard Library Source Code included with Developer's Edition
- ✓ Seamless LIM/EMS Support via new handle pointers or directly via EMS library functions.
- ✓ Full MS Mouse Library
- ✓ OS/2 Compiler Option
- ✓ 99% ANSI C Compatible
- ✓ Improved code size/speed

C++ Source Level Debugger
- ✓ Also Debugs C
- ✓ Assembler Debugging with access to registers and memory.
- ✓ 16 Debugging Windows
- ✓ Multiple Statement Lines
- ✓ Break/Trace/Watchpoints
- ✓ Dual Monitor Support
- ✓ Full C++ name unmangling for easy use
- ✓ Block memory write protect

C++ Tools Classes
- ✓ 25 C++ Classes with full source code
- ✓ Includes new Text User Interface Classes
- ✓ Event Queue, BCD Maths, Linked Lists, Money, DOS error handling classes, text windows and editing classes, virtual arrays, time and date handling, directories and filenames, interupt vectors, etc...

PRICES

C++ Compiler	$199.95
C++ Debugger	$149.95
C++ Tools	$149.95
Library Source	$149.95

Save $200 - Get the Developer's Edition for only $450 (includes all the above items).

OS/2 Option	$149.95
C++ Video	$499.95

USA: Zortech Inc.
4-C Gill Street
WOBURN MA01801
Voice: 617-646-6703
Fax: 617-643-7969

EUROPE: Zortech Ltd.
106-108 Powis Street
LONDON SE18 6LU
Voice: 44+ 81-316-7777
Fax: 44+ 81-316-4138

"ANNOUNCING V2.1"

640K Memory Barrier Smashed!

- **New** VCM™ (Virtual Code Manager) technology
- **New** Rational DOS Extender technology for compiling/ debugging massive programs
- **New** Virtual C++ Source Level Debugger requires only 4k RAM!
- **New** Remote Debugging via serial port
- **New** Powerful Environment with Browser
- **New** Completely Revised & Expanded C++ Tools
- **New** Improved Compiler Optimization

Zortech VCM™ for DOS

With Zortech's Virtual Code Manager (VCM) you can compile standard MS-DOS applications containing up to 4Mb of code. VCM is a sophisticated virtual memory system that dramatically improves performance over conventional overlay methods. Naturally, our debugger understands VCM too!

Rational™ DOS Extender Technology...

Version 2.1 incorporates this new technology for compiling and debugging really big programs on 286, 386 or 486 based PC's. You can also use V2.1 together with Rational Systems DOS Extender (purchased separately) to produce your own applications which can access memory beyond the 640k DOS limit.

C++ Debugger in 4k RAM!

Zortech's Virtual C++ Source Level Debugger can now locate itself in extended memory on 386 machines. This requires only 4K of conventional RAM!

STOP PRESS - NEWS FLASH

386 Compiler/Debugger Option (using Phar Lapp DOS Extender), UNIX 386 Compiler and OS/2 Debugger all available soon. Also new C++ Classes and Addison Wesley ZTC++ book.

ORDER/UPGRADE HOTLINE 1-800-848-8408

Circle 182 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

provided via the RPC mechanism, along with RPC interface code that may be called directly from FORTRAN. Lahey supports a long list of compatible library products, most of them real-mode 8086 code. MetaWare High C-386 may be easily linked with Lahey F77l-EM/32 modules.

The manual is excellent. It fully documents the FORTRAN syntax and contains appendices to describe compiler options, linking to assembly language, compiler and runtime limitations, numeric exceptions, and utility library calls. All extensions to ANSI FORTRAN-77 are denoted with colored text. This is the only manual that clearly documents non-ANSI extensions. The index alone is 30 pages long.

The Lahey FORTRAN compilers are all very easy to use. The compilers issue good diagnostics and useful warnings. A competent source code FORTRAN debugger (SOLD) is included. The compiler can generate symbol information stored in a separate SLD file, allowing symbolic debugging without recompilation. VAX extensions include NAMELIST, DO-ENDDO, DO WHILE-ENDDO, and a large number of intrinsic functions. F77L-EM/32 also supports VAX extensions to the OPEN statement (BLOCKSIZE) to better control I/O.

Compile times are very fast. The executable file sizes were often large since the L32 linker does not support a packed format. Benchmark runtimes were mixed with most in the mid-range of execution times. However, the speed of formatted file I/O was faster by a wide margin than any of the other compilers.

LPI FORTRAN

Language Processors (LPI), a Unix compiler vendor, has ported COBOL, FORTRAN, PL/1, BASIC, C and RPG compilers to the 80386 under Unix and Xenix. All these languages except C and RPG are also available from LPI for 386 DOS extenders. LPI FORTRAN programs can be linked with any of LPI's other languages. LPI FORTRAN supports both 80287 and 80837 inline code generation.

The Phar Lap DOS extender, linker, and librarian are bundled with the FORTRAN package (no assembler). The compiler can generate code to run under either the Phar Lap or IGC runtime. Their *init* module senses at runtime which DOS extender is operating. The compiler supports a variety of FORTRAN language extensions including DO-ENDDO, DO WHILE-ENDDO, and VAX intrinsic functions. The compiler can generate a pseudo assembly language listing using Unix (not Intel) mnemonics. The package comes with a driver program and a post processor to build LNK command files, so that a program in several hundred modules can be built with one command.

The compiler executes slowly. The optimizer needed to be partly disabled to get some test programs to execute properly. The code generated for the benchmark programs executed about average in speed compared with the other FORTRAN compilers. The DOS compiler has not been updated for over 18 months (Unix versions have been updated).

MICROWAY NDP FORTRAN

MicroWay offers C, FORTRAN, and Pascal compilers for the 386 under both DOS extenders and 386 Unix/Xenix, with mixed-language development supported. Separate linkable 386 libraries are available for FFTs, scientific numerical algorithms (NAG), and plotting. A port of the AT&T v2.0 C++ translator is also available. The compilers are written by Green Hills and enhanced by MicroWay. Intel 80287, 80387 and Weitek math chips are supported. FORTRAN-386 is compatible with Berkeley Unix FORTRAN compilers with a number of VAX extensions.

The package comes with a driver program, F77. Also included are a mini-MAKE facility and useful 386 library routines for text windows, graphics, and debug routines. A source-level debugger is soon to be available at extra cost.

FORTRAN-386 supports a large (almost bewildering) number of command-line options. The driver program could benefit from an on-line help option. The OPEN statement supports a number of VAX extensions, but many are just accepted (scanned) and not implemented. Other VAX extensions include DO-ENDDO, DO WHILE-ENDDO, and intrinsic functions. NDP FORTRAN provides one feature not present in any of the other compilers: it figures out at compile time which variables must be SAVEd, and converts these to statics. Overall, the documentation is solid, with excellent sections on numerics and optimization.

This compiler will optimize an entire function, so lots of memory for the tree structures and optimization tables is required. I would recommend a four-megabyte machine as a minimal configuration. I was unable to optimize a 2,000 line FORTRAN procedure in eight megabytes of memory, even using the Phar Lap Virtual Memory Manager.

The compiler generates assembly language files, which must then be assembled and linked with the Phar Lap tools, purchased separately. Because of this intermediate step, NDP compile/assemble times are slow and become really sluggish when optimizing large modules. However, the intermediate assembly language files do allow bit-twiddlers to hand-optimize code. The assembly language files would also be useful for taking advantage of the vectorizing features of the Weitek math chips. Overall, the NDP library is very strong on math chip control and numerics exception handling. A new compiler version that directly generates object files is imminent.

Although the underlying file I/O routines were supposedly speeded up for unformatted files, I found formatted file I/O to be slow. The default buffer sizes are 8K, but the file I/O times were still slow. Some low-level routines are available to increase buffer sizes. The numerics code generated executes fast.

OTG FTN77/386

OTG Systems markets the University of Salford FTN77/386 and DBOS DOS extender. Salford is most well-known for fine FORTRAN compilers available on Prime minicomputers. No doubt, FTN77/386 builds upon their minicomputer experience. The Salford compiler supports the 80287, 80387 and Weitek math chips. The package is bundled with a linker, librarian, source-code debugger, 386 DOS extender, on-line help programs, and other utilities.

This product is unique since it uses its own proprietary DOS extender

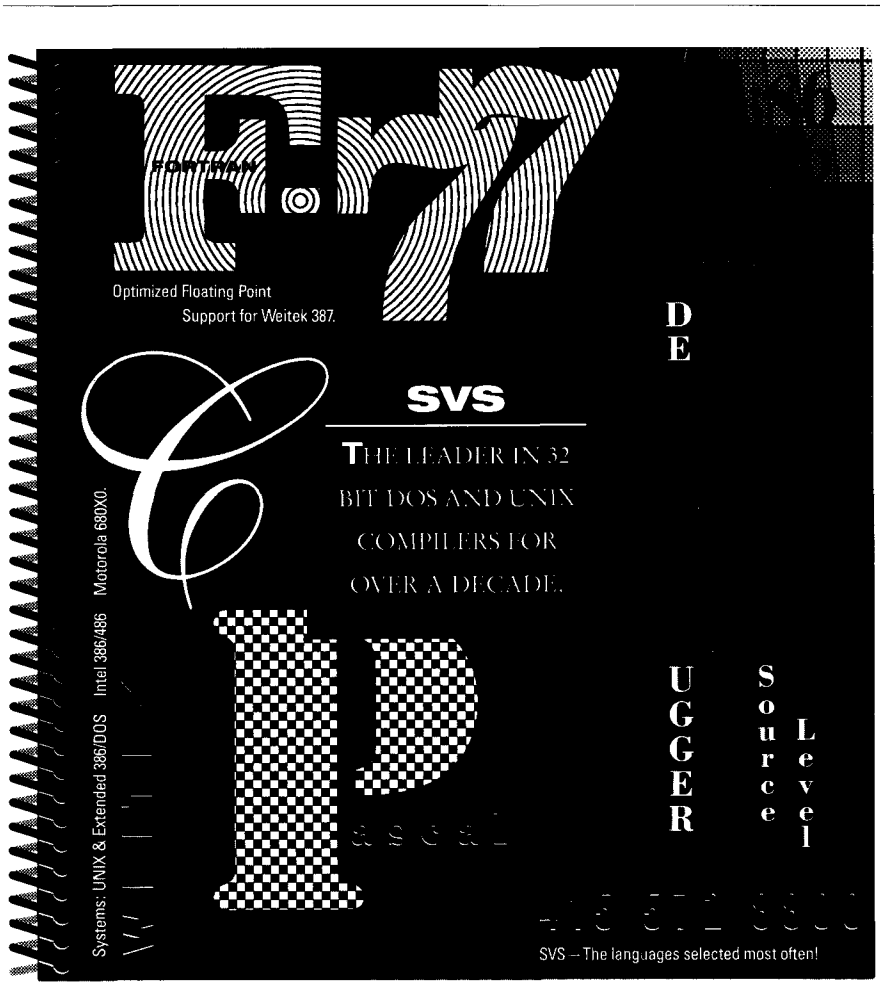
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TABLE 1: 80386 DOS FORTRAN COMPILERS

	Lahey	LPI	MicroWay	OTG	SVS	WATCOM
Product	F77L-EM/32	LPI-FORTRAN	FORTRAN-386	FTN77/386	FORTRAN-386	FORTRAN 77/386
Version	3.0	03.00	2.0.1	2.20	2.8	8.0
Base Price	$895	$795	$595	$1095	$825	$1095
386 DOS Tools:						
Linker	Y	Y	N	Y	N	Y
Extender	Y	Y	N	Y	N	N
Cost	$395	-0-	$ 495	-0-	$ 495	$ 495
Minimum Package Cost	$1290	$795	$1090	$1095	$1320	$1590

Lahey Computer Systems, P.O. Box 6091, Incline Village, NV 89450, (800) 548-4778
LPI, 959 Concord Street, Framingham, MA 01701, (508) 626-0006
MicroWay, P.O. Box 79, Kingston, MA 02364, (508) 746-7341
OTG Systems, P.O. Box 239, Clifford, PA 18413-0239, (717) 222-9100
Trio/SVS, 1710 South Amphlett Blvd., San Mateo, CA 94402, (415) 572-8800
Watcom, 415 Phillip Street, Waterloo, Ontario, Canada N2L 3X2, (519) 886-3700

DBOS (a play on Big DOS). DBOS is unusual in a number of ways. The 386 DOS extender runtime is installed as a TSR and contains much of the FORTRAN and DOS runtime libraries. DBOS supports dynamic link libraries (DLLs) and virtual memory. The DBOS DOS extender consists of a 100K executable file along with a 700K DBOS library file (DLL) containing a large number of functions. For example, the FTN77 compiler is only 24K executable, since the DBOS library is used so extensively. The linker is of similar size. All of the programs feature access to a full-screen on-line help system to ease use. The DBOS TSR may also be removed from memory.



Embedded in the FTN77 compiler is an inline 386 assembler. A large number of DOS and PC system routines are already provided. These support routines include bit-handling, storage management, character handling, screen/keyboard, time/date, file manipulation, command-line parsing, window manipulation, graphics, mouse, in-line, and miscellaneous routines—a very rich set of primitives for a FORTRAN compiler. The in-line assembler and DOS support routines are really mandatory, since DBOS is not currently supported by other 386 development tools (other compilers are to follow).

The defaults for the compiler are easily configured, and the compiler supports a load and go option for quick testing. A number of FORTRAN options provide useful debugging aids. These aids include runtime checking of array subscripts, arithmetic overflow, argument consistency checking, and assuring that variables and array elements have been given a value before being used. A runtime-error can pop you into the debugger, and a profile option will provide counts of how often a function was called. Compile speeds are very fast, but the execution speed of generated code was only average. The LOGICAL*1 and INTEGER*1 data types are not supported, which may

Circle 7 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

cause problems with older FORTRAN code. I found through trial and error that the debugger does not currently support debugging code for the Weitek chip. File I/O buffer sizes are too small, and no extensions are incorporated for easily changing buffer sizes. Executable file sizes are very small, since part of the application code is in the DBOS extender library. Despite these shortcomings, the speed of development made this product a pleasure to use.

SVS FORTRAN-386

Trio/SVS markets FORTRAN, Pascal, and C compilers for both 386 DOS extenders and versions of Unix/386. SVS compilers are also available on a number of 68020/030 Unix workstations. SVS compilers for the 386 were originally marketed by Science Applications (SAIC) and used the IGC DOS extender. Trio/SVS has now ported their compilers to the Phar Lap DOS extender and markets them directly. SVS FORTRAN-386 supports a number of VAX and IBM extensions including NAMELIST, ENCODE/DECODE, DO-ENDDO, DO WHILE-ENDDO, and a number of intrinsic functions.

The compiler generates in-line code for either the 80287, 80387, or Weitek math chips and object files in the Phar Lap EASY-OMF format. The user must purchase the Phar Lap Developer's Kit (386 linker) in order to create executable programs with the SVS compilers. The Weitek option is an additional $170. Mixed-language development in FORTRAN, C, and Pascal is supported.

The SVS source code symbolic debugger (DBG) is an extra option ($325). Separate DBG files are generated as a compiler option, so debugging does not require recompilation. SVS DBG is a simple command-driven debugger that allows setting break and trace points, displaying source code, displaying and setting variables and data structures. A calling sequence backtrace can display active subroutines and memory can be displayed and changed.

The SVS languages come with a compiler driver program designed to automate the compile and subsequent link steps. The SVS compilers go through a three-step compile (front-end language processor, code generation/optimization, and object file formatter). Even executing these three separate programs for each source file, the SVS compilers execute very fast.

SVS allocates common blocks and variables on the stack, unless told otherwise (via SAVE). Without the SAVE option, the first invocation of a program may crash the Phar Lap runtime until the stack size is set properly for your application. This points to a general weakness in SVS stack handling. The Phar Lap linker places segments into memory in the order defined. SVS places CODE lowest in memory, followed by the STACK and finally the DATA segments. Both DATA and STACK segments use the same selector, which also overlays the CODE segment. This allows easy reading and writing to the CODE segment for DOS junkies, but also lets your stack underflow over your CODE. SVS needs to revise their stack handling (set the stack at the top of memory or set limits on a separate STACK segmentor selector) and will likely do so soon.

The compiler issues useful diagnostics and will warn if parameters are type-mismatched between calling and called procedures. Compile times are fast. The benchmark execution times are generally faster than the other 80386 compilers using both the 80837 and Weitek. The buffers for file I/O are defaulted too small (1K), and no (VAX) language extensions are provided to increase buffer size, so expect file I/O to be slow. Numeric exception handling is not as strong as that provided by Microway or Lahey. These are clearly areas for improvement.

WATCOM FORTRAN/386

I also worked with a beta test version of Watcom FORTRAN and will offer some early observations. FORTRAN/386 comes with an editor, compiler, linker, librarian, windowing source-level debugger, profiler, code disassembler, and other utilities. This provides a very complete development environment. The Watcom compiler supports a large set of FORTRAN language extensions, and Watcom FORTRAN/386 will generate code for either the Phar Lap or Ergo DOS extenders. Watcom C and

FORTRAN

OVER 70% OF OUR NEW BUSINESS COMES FROM SATISFIED CLIENT REFERENCES!

FTN77/386 & 486 POWER COMPILERS

FTN/386 & 486 uses DOS extender technology to turn your 386 & 486 based PC into a powerful engineering workstation. With FTN77/386 & 486, your programs run under MS-DOS, but have a 4 GIGABYTE address space and super-mini execution speed. FTN77/386 is a full ANSI 77 compiler validated by the US Dept. of Commerce, with many extensions for mainframe and DOS compatibility. A window based source level debugger and profiler is included. Compilation is blazingly fast, and compile and run-time diagnostics are the best available. **Weitek now supported. Interface to Desqview.**

Developed by: Salford Software Marketing, LTD

FORTRAN RESTRUCTURING AND MAINTENANCE TOOLKIT

A pass through PRISM will rejuvenate your old FORTRAN programs. PRISM analyzes the bewildering jumble of GOTOs, arithmetic IFs etc. which are characteristic of old-style FORTRAN, and re-writes the program using modern structured programming constructs. Output from PRISM is neatly indented and structured but, amazingly, is exactly equivalent logically to the input spaghetti code.

- Restructures up to 5,000 LPM
- Available for IBM, VAX, SUN, HP, Prime and others.
- Each version extended to deal with Local FORTRAN dialect.
- Fast and easy to use. On a VAX, just type "SPAG *.FOR".

Developed by: Polyhedron Software, LTD

FORTRAN TO "C" TRANSLATOR

PROMULA.FORTRAN ADDS VALUE TO YOUR CODE DURING TRANSLATION

- Compatibility under C, produces the same results as the original code.
- Maintainable C source code which the original author can support.
- Virtural Memory Logic Supported.
- Dynamic Memory Logic Supported.
- References to external Data Bases.
- FORTRAN66 and FORTRAN77 mixed dialect Supported.

PROMULA.FORTRAN provides portability within the C world, extensive control over the translation, easy readability for maintenance and support. Available for DEC/VAX, SUN/UNIX, Prime/Primos, 386/MS-DOS

Developed by: Promula Development Corporation

100% Satisfaction Guaranteed.

OTG SYSTEMS, INC.

Distributed exclusively by:
OTG SYSTEMS, INC.
Suite 300, Rts. 106 & 374
P.O. Box 239
Clifford, PA 18413-0239

Tele 717.222.9100 - FAX 717.222.9103

Circle **270** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TABLE 2: BENCHMARK EXECUTION AND COMPILE TIMES

	Lahey	LPI	NDP	OTG	SVS	WATCOM	VAX 785
Whetstones (K/sec)							
Single(387)	2,024	2,092	2,426	2,045	2,560	2,020	1,751
Double(387)	1,760	1,720	2,222	1,896	2,320	1,840	1,183
Linpack (MFLOPS/sec)							
Single(387)	0.29	0.21	0.31	0.26	0.31	0.30	0.35
(Weitek)	0.52	N/A	0.66	0.52	0.70	N/A	N/A
Double(387)	0.24	0.17	0.26	0.22	0.25	0.24	0.14
(Weitek)	0.35	N/A	0.37	0.34	0.49	N/A	N/A
Runtimes (min:secs)							
Superlite(387)	2:05	2:07	2:09	2:20	1:58	2:17	1:53
(Weitek)	(1)*	N/A	(2)**	(1)*	0:51	N/A	N/A
Chilc	5:51	6:00	6:00	6:02	5:28	5:47	8:15
(Weitek)	3:32	N/A	3:20	4:05	2:46	N/A	N/A
Update	0:55	2:46	1:40	2:39	2:15	2:25	
Compile/Link Times (min:secs)							
Superlite(387)	0:40	2:51	3:28	0:17	0:40	7:36	
Update	0:23	2:02	1:36	0:12	0:23	1:02	

NOTES: Runtimes on Everex Step 386-25 with 80387 and WTL 3167
*(1) Program crashed
**(2) Ran out of memory compiling in 8 megabytes

People are talking about us.

F77L-EM/32

Port 4GB mainframe programs to 80386s with this 32-bit DOS-Extender compiler. The Winner of *PC Magazine's* 1988 Technical Excellence Award just got better. New Version 3.0 and OS include: Editor, Make Utility, Virtual Memory Support, DESQview Support, New Documentation and Free Unlimited Runtime Licenses. F77L-EM/32 $895 OS/386 $395

F77L

The compiler of choice among reviewers and professionals. Includes a Debugger, Editor, Profiler, Linker, Make Utility, Weitek and 386 Real-Mode Support, Graphics. $595

Lahey Personal Fortran 77

New Version 3.0: Full ANSI 77, Debugger, Editor, Linker, Library Manager, Microsoft and Borland C interfaces, 400 page Manual, Unbeatable Price. $99

When people talk about FORTRAN the name mentioned most often is Lahey Computer Systems Inc

Contact us to discuss our products and your needs. **(800) 548-4778**
Lahey Computer Systems, Inc. P.O. Box 6091, Incline Village, NV 89450
Tel: (702) 831-2500 FAX: (702) 831-8123 Tlx: 9102401256

FORTRAN IS OUR FORTE

FORTRAN code can be easily linked. However, only the 80387 math coprocessor is supported.

The profiler and windowing (Codeview style) debugger will make this an attractive product. The compiler runs on the slow side especially compiling large modules, but the numerics code executes better than average in speed. File I/O is slow, partly a result of small buffer sizes. These are all areas likely to be addressed in the official release.

BENCHMARKING

For testing purposes, I ran compiles and execution benchmarks in the same hardware—an Everex Step 386-25 with 8 megabytes of 32-bit memory and a fast ESDI hard disk. The Everex machine was selected because it is about the fastest 80386 machine at this clock rate. Program execution times are very sensitive to memory and RAM cache design.

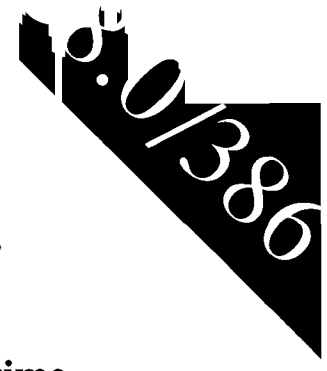
I used the same numerics and file I/O intensive programs for benchmarking as in the past (see *PJ* 6.6). I don't put much value in the Whetstone benchmark. However, the LINPACK programs are good measurements of well-written code executing matrix manipulations. The SUPERLITE benchmark is a ray tracing routine that calculates daylight in a space—all single precision with lots of transcendentals. The CHILC program is all double precision with few transcendentals. The UPDATE program emulates the CDC UPDATE utility and tests file I/O (sequential formatted).

A fast 25 MHz 80386/80387 is clearly competitive with VAX 785 execution times. For double precision, the microprocessors appear superior. The Weitek 3167 chip in a 25 MHz 80836 makes a dramatic improvement.

Part of these execution speed improvements disappear because of the typically slow I/O delivered by the 386 FORTRAN libraries; consider that the best time for FORTRAN formatted I/O with Lahey F77L-EM/32 was still 60% slower than running the same benchmark with Microsoft FORTRAN 5.0 (with 16K BLOCKSIZEs) under DOS. Clearly, all of these FORTRAN vendors need to work on file I/O in their libraries for 386 DOS extenders as well as on code optimization. PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Unleash 386 Power on Your Microsoft C Code.



- ☑ Source-level debugger
- ☑ Graphics Library
- ☑ Fast, tight code
- ☑ Generates high-performance code for 32-bit protected mode
- ☑ Microsoft source and library compatible
- ☑ Supports Phar Lap and Eclipse DOS Extenders
- ☑ Protected-mode version of compiler
- ☑ SAA Compatible
- ☑ Run-time Compatible with WATCOM FORTRAN 77 /386
- ☑ 100% ANSI C Compatible
- ☑ Profiler
- ☑ Remote debugging

Experts Agree on WATCOM C:

"When Novell went looking for a 32-bit compiler for use with the NetWare 386 developer's kit, the company selected WATCOM's... It's clear that Novell chose wisely; this product is a winner."
Fred Hommel
BYTE, December 1989

"WATCOM C broke with tradition to make a fast, efficient C programming environment that has other C compiler designers rethinking their strategy."
Bill Machrone
PC Magazine, January 17, 1989

"Good things do get better."
John Dlugosz
Dr. Dobbs Journal, September 1989

Microsoft library- and source-compatibility makes WATCOM C7.0 /386 ideal for porting DOS applications to 32-bit native mode. This compiler enables full 386 performance without 640K limitations.
Richard M. Smith, President
Phar Lap Software, Inc.

"WATCOM's latest release of its rising star is a clear winner...It is a clearly superior value when compared to all compilers..."
Richard Relph
INFOWORLD, May 22, 1989

"...WATCOM C showed shining performance."
Computer Language, Feb. 1989

Also announcing:

WATCOM FORTRAN 77 /386

- Based on WATCOM C Technology
- Shares WATCOM C Development Tools
- Supports FULL ANSI FORTRAN 77 Language plus Extensions
- SAA Compatible
- Generates High-performance Code for 32-bit 386 Native Mode

WATCOM C8.0 /386

- 100% ANSI C Optimizing Compiler
- 386 Run-time Library Object Code
- 386 Graphics Library
- Windowed Source Level Debugger
- Profiler, Editor, MAKE and Linker
- Object-code Librarian
- Object-code Disassembler
- Protected-mode version of compiler
- Supports Phar Lap and Eclipse DOS extenders

WATCOM F77/386

- 100% ANSI FORTRAN 77 Optimizing Compiler plus extensions
- 386 Run-time Library Object Code
- Windowed Source-level Debugger
- Profiler, Editor, MAKE and Linker
- Object-code Librarian
- Object-code Disassembler
- Protected-mode version of compiler
- Run-time compatible with WATCOM C8.0 /386

1-800-265-4555

WATCOM

WATCOM, 415 Phillip Street, Waterloo, Ontario, Canada N2L 3X2, Telephone: (519) 886-3700, Fax: (519) 747-4971
Trademark/Owner: Phar Lap/Phar Lap, Inc.; Microsoft/Microsoft, Inc.; Eclipse/Eclipse Computer Solutions, Inc.
WATCOM & Express C are trademarks of WATCOM Systems Inc.
© Copyright 1990 WATCOM Products Inc.

Contact this advertiser directly.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A stack data type implemented with the preprocessor

Using C++ features in C

By Allen Holub

C++ is often billed as the language of the future, but mostly by unobjective (so to speak) sources. Though C++ has some nice features, like true abstract data types, you can actually use C to do a lot of what C++ does. An *abstract type* is a high-level type that is used to manipulate objects of a more basic type. In this article, I'll look at a "stack" abstract type that's implemented entirely with the preprocessor. Stacks can be made up of most of the basic types in C (stack of ints, character pointers, etc.)

Explanations of the stack macros are given in the code accompanying this article, and the stack macros are implemented in STACK.H (Listing 1). These macros make heavy use of the ANSI concatenation operator, "##". This operator is removed from a macro definition by the preprocessor, effectively concatenating the strings on either side of it. For example, given the invocation

```
stack_dcl(plates,int,128);
```

the typedef on line 7 of Listing 1 evaluates first to

```
typedef int t_##plates;
```

(the "int" replaces "type" in the definition; the "plates" replaces "stack") and then the "##" is removed, yielding:

```
typedef int t_plates;
```

Lines 8 and 9 will evaluate as follows:

```
t_plates plates[128];
t_plates *(p_plates) = plates + (128);
```

given the earlier **stack_dcl( )** invocation. A downward-growing stack is created, and so the stack pointer (*p_plates*) is initialized to just past the end of the stack. This initialization works correctly in ANSI C, which says that pointers can point one cell beyond the end of an array and still be valid, but a pointer is not permitted to go to the left of the array.

Since a downward-growing stack is implemented, a push is a predecrement (on line 25), and a pop is a postincrement (on line 33). This way the stack pointer will always be pointing to the top-of-stack item. The actual push and pop operations have been isolated into the **push_( )** and **pop_( )** macros because it's occasionally useful to do a push or pop without checking for stack overflow or underflow. Note that the **popn_( )** macro on line 36 must evaluate to the top-of-stack element before anything is popped. A postincrement is simulated here by modifying the stack pointer by *amt* elements, then going backward with a *[-amt]* to get the previous top-of-stack element.

The **stack_err( )** macro on lines 41 and 42 prints an error message and evaluates to 0. You can check the returned value from **push( )** or **pop( )** to test this error value. You should change it to something other than 0 if 0 is a legitimate value to push or pop. You need two casts on lines 29 and 33 where **stack_err( )** is invoked because there's no telling to what sort of value the macro will actually evaluate. The default definition evaluates to an int, but since **stack_err( )** can be redefined, you can't count on that. The stack type is also uncertain because it can be redefined.

You can't just cast **stack_err( )** to the same type as the stack because the stack might be an array of pointers, and many compilers complain if you try to convert an int to a pointer. So you have to cast twice—first to long and then to the stack type—to suppress this error message. Though casting to long as an intermediate step fixes this problem, it might introduce an unnecessary conversion if the stack type is an int, short, or char. These multiple type conversions will also cause some portability problems if the **stack_err( )** macro evaluates to something that won't fit into a long (like a double).

The earlier macros are fast, but they have several problems, the main one being inflexibility. You'd like the stacks to grow

---

Allen Holub is a teacher, writer, independent consultant, and composer. He teaches C and compiler design for the University of California at Berkeley Extension and has authored several books, most recently Compiler Design in C. *You can contact him at holub@violet.berkeley.edu (...!ucbvax!violet!holub) via USENET or c/o Software Engineering Consultants, P.O. Box 5679, Berkeley, CA 94705 by US snail. His scotch is not Dewers.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

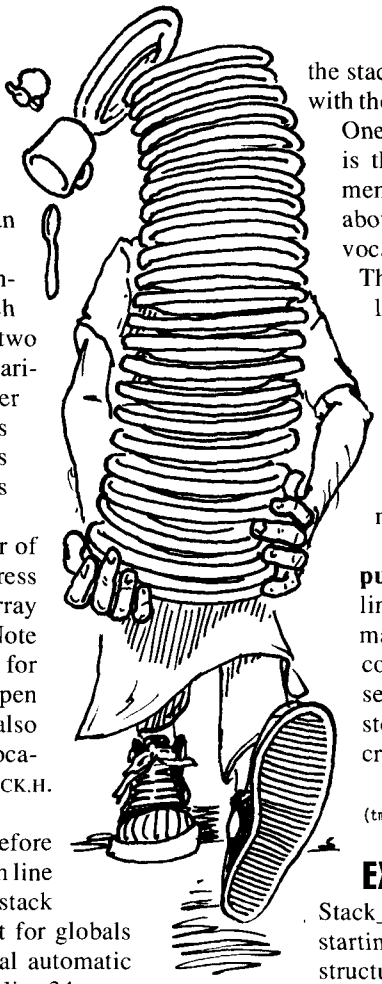
in size as items were pushed, since you'd never run out of space. The stack macros are rewritten in Listing 3 for this capability. These macros have the same interface as the earlier macros, so they're interchangeable—you should just include EXPSTACK.H rather than STACK.H at the top of your program.

I've taken a completely different implementation approach here—one that is much closer to a C++ class. A **stack_dcl** declares two objects. The first is an int-sized temporary variable that will be used by one of the other macros. The second is a structure that contains all relevant information about the stack. (It's declared on lines 23 to 30). This structure is the rough equivalent of a C++ object.

The structure has three fields: the number of elements in the stack array (*sz*), the base address of the array (*base*), the address of the last array element (*top*), and the stack pointer (*sp*). Note that no memory is allocated by a **stack_dcl( )** for the stack itself. Memory allocation will happen on the first push. The **stack_dcl( )** macro is also passed a third argument (size) so that the invocation syntax is identical to the macro in STACK.H. This third argument is ignored, however.

The stack structure must be initialized before it's used by an explicit call to **stack_init( )** (on line 32). The macro zeros out the memory in the stack structure, so you don't really have to call it for globals and local statics. You need the call for local automatic stacks, however.The **stack_clear( )** macro on line 34 complements the initialization macro. This version of **stack_clear( )** doesn't just reset the stack pointer, as was the case in the earlier version; instead, it frees all memory associated with the stack and then zeros the entire stack structure. The conditional on line 32 makes sure that the memory exists before freeing it.

The **stack_clear( )** macro can be called anywhere to empty the stack, but it must be called when you're finished with the stack in order to free any associated memory.



One real advantage that C++ has over straight C is that you can automate the initialization and memory deallocation process rather than worrying about explicit **stack_init( )** and **stack_clear( )** invocations.

The macros on lines 37 to 46 work pretty much like the earlier versions. There are two differences, though. First, the stack grows when it gets full, so **stack_full( )** always evaluates to 0. Second, the stacks now grow up rather than down—a preincrement is used for a push and a postdecrement for a pop. Consequently, **stack_item( )** and **stack_ele( )** are modified to handle the new situation.

The killer macros are invoked indirectly by **push( )** on line 59, which invokes **stack_gr( )** on lines 48 to 58 when the stack has to grow. The macro is actually several independent statements connected with comma operators rather than semicolons. On line 48, **stack_gr( )** starts out by storing the current stack size in the variable created by the **stack_dcl( )**:

```
(tmp_##stack = stack_ele(stack)),
```

EXPANDING STACKS

Stack_gr( ) now proceeds to expand the stack size, starting by modifying the size component of the structure:

```
(stack.sz += stack_chunk),
```

The next five lines reallocate the stack space. If *stack.base* is not NULL, then **realloc( )** is used. Otherwise, a newly initialized or newly cleared stack is being accessed for the first time, in which case a simple **malloc( )** is used:

LISTING 1

```
1  /* Stack.h     Stack-maintenance macros. Creates downward-growing
2   * stacks          (which should work in all six memory models).
3   */
4
5  #define stack_cls    /* empty */
6
7  #define stack_dcl(stack,type,size)  typedef type t_##stack; \
8                                      stack_cls t_##stack stack[size];\
9                                      stack_cls t_##stack (*p_##stack)\
10                                                 = stack + (size)
11 #define stack_clear(stack) (  (p_##stack)  = (stack + \
12                               sizeof(stack)/sizeof(*stack)) )
13
14 #define stack_init(s)      stack_clear(stack)
15
16 #define stack_full(stack)  (  (p_##stack) <= stack )
17 #define stack_empty(stack) (  (p_##stack) >= (stack + \
18                               sizeof(stack)/sizeof(*stack)) )
19 /* stack_ele evaluates to the number of elements on the stack */
20 #define stack_ele(stack) ((sizeof(stack)/sizeof(*stack))- \
21                          (p_##stack-stack))
22 /* stack_item evaluates to the item at offset from top of stack */
23 #define stack_item(stack,offset) ( *(p_##stack + (offset)) )
24 #define stack_p(stack)  p_##stack
25 #define push_(stack,x)  ( *--p_##stack = (x) )
26 #define pop_(stack)     ( *p_##stack++      )
27 /* push_ and pop_ do not perform checks for under/overflow */
28 #define push(stack,x)   ( stack_full(stack) \
29                                 ? ((t_##stack)(long)(stack_err(1)))\
30                                 : push_(stack,x)                   )
31
32 #define pop(stack)      ( stack_empty(stack) \
33                                 ? ((t_##stack)(long)(stack_err(0)))\
34                                 : pop_(stack)                      )
35 /* popn and popn_ pop amt items from the stack */
36 #define popn_(stack,amt) ( (p_##stack += amt)[-amt]  )
37 #define popn( stack,amt) ( (stack_ele(stack) < amt)                \
38                                 ? ((t_##stack)(long)(stack_err(0))) \
39                                 : popn_(stack,amt)                 )
40 /* stack_err is called from macros, may of course be redefined */
41 #define stack_err(o) ((o) ? fprintf(stderr,"Stack overflow\n" ), 0 \
42                           : fprintf(stderr,"Stack underflow\n"), 0 )
```

LISTING 2

```
1  #include <stdio.h>
2  #include "stack.h"
3
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Program FASTER with ...

CC-RIDER

THE C PROGRAMMER'S COMPANION

NEW!

PROFESSIONAL EDITION

CROSS-REF Editing!

Instantly recall your C symbol definitions by hitting a *hot-key* inside any editor! Along with standard CCRIDER features like Source View, Edit and Definition Paste, the new *Professional Edition* has a Source Browse mode that lets you walk through all *usages* of a symbol in your code for ***on-line hypertext cross-referencing as you edit your program!***

Function Prototypes!

Build a database of useful information for your C code with CCSYM, a powerful ***source code analyzer*** which generates a wealth of useful output, like fully commented static and global function prototype #include files and a detailed log file documenting all your symbols plus where and how they're used. Uses expanded memory for ***fast*** source code analysis!

QuickHelp, too!

CCSYM can also create a ***help database for your program*** which is compatible with Microsoft's help system, *QuickHelp*. All symbols in your application are accessible from QuickHelp's menus, with ***documentation extracted automatically from your original source code.***

Works with ANSI C, plus Microsoft/Zortech/Borland C.

	Standard	*Professional*
	$89 / 179 DOS / OS2	***$279*** DOS & OS2

Western Wares

(303) 327-4898

Box C Norwood, CO 81423

FREE DEMO DISK

*Circle **100** on Reader Service Card*

New and Improved

TRANSLATOR 3.0
(Pascal to C)

- Now, even faster and more efficient.
- Translates Jensen and Wirth Pascal to ANSI C. Use with your Turbo Pascal.
- Provides interactive display while the translation proceeds.
- Includes the fastest search routines to provide super fast translations.
- Still many examples, explaining the fine points, in the User's Manual.

Still only
$99.95

(or upgrade from 2.X for $49.95)

MILTON BROWN SOFTWARE
935 Chester Avenue
San Marino, CA 91108
(818) 796-5527

```
 4  main()
 5  {
 6      int i;
 7
 8      stack_dcl  ( stack, int, 20 );
 9      stack_init ( stack );
10
11      for( i = 0; i <= 10; ++i )
12          push( stack, i );
13
14      while( !stack_empty(stack) )
15          printf("%d\n", pop( stack ) );
16
17      stack_clear ( stack );      /* free all memory */
18  }
```

LISTING 3

```
 1  /* ExpStack.h   Expandable-Stack maintenance macros. Creates
 2   * downward-growing stacks.
 3   * These stacks automatically scale as needed. You MUST call
 4   * stack_init for all nonglobal stacks before using any of the other
 5   * stack macros, however. You must call stack_clear when you're done
 6   * with the stack (before you leave the scope of the stack_dcl).
 7   *
 8   * Stacks grow up.
 9   *
10   * There's no popn() or popn_(), I leave it as an exercise to the
11   * reader to add them.
12   */
13
14  extern void *realloc ( void *, unsigned int );
15  extern void *malloc  ( unsigned int );
16
17  #define stack_chunk     32      /* Stack grows this many elements
18                                     at a time */
19  #define stack_cls               /* empty */
20
21  #define stack_dcl(stack,type,x) typedef type t_##stack;
22                                  stack_cls int tmp_##stack;\
23                                  stack_cls struct stack\
24                                  {\
25                                      int sz;\
26                                      t_##stack *base;\
27                                      t_##stack *top;\
28                                      t_##stack *sp;\
29                                  }\
30                                  stack
31
32  #define stack_init(stack)       memset( stack, 0, sizeof(stack) )
33
34  #define stack_clear(stack)      (stack.base ? free(stack.base) : 0),\
35                                   memset( stack, 0, sizeof(stack) )
36
37  #define stack_ele(stack)        (!stack.sz ? 0 : (stack.sp \
38                                  -stack.base) +1)
39  #define stack_full(stack)       0
40  #define stack_empty(stack)      (stack_ele(stack) == 0)
41
42  #define stack_item(stack,offset)    ( *(stack.sp - (offset)) )
43  #define stack_p(stack)          ( stack.sp )
44
45  #define push_(stack,x)          ( *++stack.sp = (x)     )
46  #define pop_(stack)             ( *stack.sp--           )
47
48  #define stack_gr(stack) (tmp_##stack = stack_ele(stack)),\
49                          (stack.sz   += stack_chunk),\
50                          (stack.base  = stack.base \
51                              ? ((t_##stack *)realloc(stack.base,\
52                                      stack.sz * sizeof(t_##stack)))\
53                              : ((t_##stack *)malloc(\
54                                      stack.sz * sizeof(t_##stack)))\
55                          ),\
56                          (stack.base ? 1 : stack_err(1)),\
57                          (stack.top = stack.base + (stack.sz - 1)),\
58                          (stack.sp  = stack.base + (tmp_##stack - 1) )
59  #define push(stack,x)   ( ((!stack.sz || stack.sp >= stack.top) \
60                              ? stack_gr(stack) : 0),\
61                            push_(stack,x) )
62  #define pop(stack)      ( stack_empty(stack)\
63                                ? ((t_##stack)(long)(stack_err(0)))\
64                                : pop_(stack))
65  #define stack_err(o) ((o)? (fprintf(stderr,"Out of Memory!\n"),\
66                          exit(1))
67                          : (fprintf(stderr,\
68                             "Stack underflow\n"), exit(1))     )
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C CODE FOR THE PC

source code, of course

NEW!	Network Program-to-Program Communication (message service and IPC based on NETBIOS; pure assembler with C bindings)	$600
	C-Data Manager (object-oriented data management, persistent objects from runtime definitions, network and entity models)	$495
	GraphiC 5.0 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
	Pinnacle (relational DBMS with debugging calls and sparse table & repeated field support)	$325
NEW!	Time Series Management Tool (collection of routines for manipulating time series)	$250
	TurboTEX (Release 2.0; HP, PS, dot drivers; CM fonts; LaTEX; MetaFont)	$250
NEW!	Time Series Management Tool (collection of routines for manipulating time series)	$250
	db_File & db_Retrieve by Raima (B-tree and network database with SQL query and report writer; multi-user $475)	$245
	Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF; specify compiler)	$225
	CDirect (multi-user hashed file manager; variable length fields, binary or ASCII data, alternate keys)	$210
	SilverComm (complete asynchronous communications library)	$210
	Install 2.3 (automatic installation program; user-selected partial installation; CRC checking)	$195
	Wendin-DOS Plus (self-bootable, multitasking, multiuser MS-DOS replacement; includes XTC editor)	$180
	QuickGeometry Library (large collection of mathematics, graphics, display & DXF subroutines for CAD/CAM/CAE/CNC)	$170
	CBTree (B+tree ISAM driver, multiple variable-length keys)	$165
Updated!	Turbo G Graphics Library (all popular adapters, hidden line removal)	$165
	TurboGeometry (library of routines for computational geometry, Version 3.0)	$160
	WKS Library Version 2.01 (C program interface to Lotus 1-2-3, dBase, Supercalc 4, Quatro, & Clipper)	$155
NEW!	C Generator (generates C code to read & write file records defined with C structure syntax)	$150
	Cephes Mathematical Library (over 100 high-quality, double-precision scientific functions)	$150
	ME Version 2.1 (programmer's editor with C-like macro language by Magma Software)	$140
	Vmem/C (virtual memory manager; least-recently used pager; dynamic expansion of swap file)	$140
NEW!	C++ Object Library (virtual windows, I/O, lists, file free space management, keyed ISAM file I/O)	$125
C++	Rogue Wave Vector & Matrix Classes (inc. C++ overloadings for standard operators, matrix inversion & FFT; Zortech or GNU C++)	$125
	Power Search by Blaise Computing (regular-expression compiler; generates machine code on the fly)	$120
	TE Editor Developer's Kit (full screen editor, undo command, multiple windows)	$115
	PowerSTOR (upto a gigabyte of heap space on extended memory, expanded memory, and/or hard disk)	$110
NEW!	Hold Everything (spawn new programs; swap parent to EMS or disk; handles video, interrupts, & environment; returns error level)	$105
	B-Strings (dynamic string handling; cut, copy, paste, search, user input, etc.; non-fragmenting memory management)	$105
NEW!	The Input Processor (recognize patterns in program input using Aho & Corasick string matching algorithm)	$105
	Minix Operating System (Version 1.3; U**x-like operating system, includes manual; specify AT or XT)	each $105
	PC/IP (CMU/MIT TCP/IP for PCs; Ethernet, Appletalk & NETBIOS drivers, RVD, gateways)	$100
	B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$100
NEW!	Disassembler (8086/80286/80386/80C186 with 80287/80387/80C187 NPX)	$90
	QC88 C compiler (ASM output, small model, no longs, floats or bit fields, 80+ function library)	$90
	Booter Toolkit (floppy disk bootstrap routines, DOS file system, light-weight multitasking, windows, fast memory management)	$85
	JATE Async Terminal Emulator (includes file transfer and menu subsystem)	$80
	Otter 1.0 (beautiful theorem-prover by Bill McCune; includes manual & two books by Wos; complete starter kit)	$80
	MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
	HY-PHEN-EX (a hyphenator for American English with over 4,800 rules)	$75
	Make (macros, all languages, built-in rules)	$75
	eval() (C function to evaluate ASCII infix expression string; 17 built-in functions)	$75
	XT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for XTs)	$75
	Symtab/Ptree (general-purpose symbol table/parse tree construction and management package; specify Symtab or Ptree)	each $75
NEW!	XSPAWN (swap parent process to expanded memory or disk before running child)	$70
	Professional C Windows (lean & mean window and keyboard handler)	$70
	Heap Expander (virtual memory manager using expanded memory, extended memory, and disk space)	$65
NEW!	Foundations-1 C++ Class Library (ASCII record I/O, bit arrays, exception handling, B-tree, persistent objects)	$60
	Quincy (interactive C interpreter)	$60
	Coder's Prolog (Version 3.0; inference engine for use with C programs)	$60
	Async-Termio (Unix V compatible serial interface for MS-DOS; stty, ioctl, SIGINT, etc.)	$55
Upgrade!	Pascal P-Code Compiler & Interpreter or Pascal-to-C Translator (Level 0 ISO standard Pascal with some Level 1 features)	each $50
	Backup & Restore Utility by Blake McBride (multiple volumes, file compression & encryption)	$50
	Floppy TAR (TAR backup and restore on MS-DOS devices; direct access to non-standard devices)	$50
	SuperGrep (exceptionally fast, revolutionary text searching algorithm; also searches sub-directories)	$50
	OBJASM (convert .obj files to .asm files; output is MASM compatible)	$50
	CLIPS (rule-based expert system generator, Version 4.3; advanced manuals available)	$50
	PCHRT (40 functions to manage multiple microsecond timers; generate precision delays; insert timers on any interrupt)	$45
	Kier DateLib (all kinds of date manipulation; translation, validation, formatting, & arithmetic)	$45
	BOLS Data Utilities (Sort/Merge/Select; five easy-to-use ASCII file management routines)	$40
	DES Encryption & Decryption (2500 bits/second on 4.77 MHz PC for on-the-fly encryption at 2400 baud)	$40
	FlexList (doubly-linked lists of arbitrary data with multiple access methods; specify C or C++)	each $40
	Virtual Memory Manager by Blake McBride (LRU pager, dynamic swap file, image save/restore)	$40
	Heap I/O (treat all or part of a disk file as heap storage)	$40
C++	NIH Class Library (collection of basic C++ classes by Keith Gorlen; V2.204 (V3.0 Pre-release); uses C++ R2.0)	$35
	Bison & BYACC (YACC workalike parser generators; documentation; no restrictions on use of BYACC output)	$35
	PC-XINU (Comer's XINU operating system for PC)	$35
	RXC & EGREP (Regular Expression Compiler and Pattern Matching; RXC makes finite state machine from regular expression)	$35
NEW!	CALC_MAT (input, output & in-core handling of variable dimension matrices of floats and text)	$30
	REGX Plus (search and replace string manipulation routines based on regular expressions)	$30
	GNU Awk & Diff for PC (both programs in one package)	$30
	6-Pack of Editors (baker's half-dozen public domain editors for use, study & hacking; includes microEmacs 3.10 & Stevie, a vi clone)	$30
	Crunch Pack (14 file compression & expansion programs)	$30
	PC-MAIL (UUCP mailer by Wietse Z. Venema; send, receive, and manage UUCP mail)	$25
Updated!	FLEX (fast lexical analyzer generator; new, improved LEX; official BSD Version 2.2 with docs)	$25
	List-Pac (C functions for lists, stacks, and queues)	$25
C++	Using C++ Library (the code from the book by Bruce Eckel and then some; Zortech 2.0 compatible)	$25
	A68 (68000 cross-assembler)	$20
	XLT Macro Processor (general purpose text translator)	$20
	Data	
	Moby Part-of-Speech (200,000 words and phrases described by prioritized part(s)-of-speech)	$120
	Moby Hyphenator (150,000 words fully hyphenated/syllabified)	$105
	Moby Words (500,000 words & phrases, 9,000 stars, 15,000 names)	$80
	Smithsonian Astronomical Observatory Subset (right ascension, declination, & magnitude of 258,997 stars)	$60
	U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
	The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
	KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	$30
	Interactive Computer Ephemerii (high-precision moon, sun, planet & star positions; USNO (no source) & Downey 4.8 (C source))	$30

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409 USA

Voice: (512) 258-0785
FAX: (512) 258-1342
E-mail: info@acw.com

Free surface shipping for cash in advance **For delivery in Texas add 7%** **MasterCard/VISA**

Circle **30** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
(stack.base  = stack.base \
 ? ((t_##stack *)realloc(stack.base,\
              stack.sz * sizeof(t_##stack)))\
 : ((t_##stack *)malloc(\
              stack.sz * sizeof(t_##stack)))\
)
```

The next line checks to see if the allocation worked or not:

```
(stack.base ? 1 : stack_err(1)),
```

If *stack.base* is zero,it means that the **malloc/realloc** call failed and **stack_err** is executed; otherwise, the macro does nothing. (That's the "1;" you need it because the **else** clause of a conditional operator is not optional, as it is in an **if** statement). The rest of the macro updates the end-of-array marker (*stack.top*) and the stack pointer (*stack.sp*):

```
(stack.top = stack.base + (stack.sz - 1)    ),
(stack.sp  = stack.base + (tmp_##stack - 1) )
```

This is a pretty big macro, and is a likely candidate for moving into a subroutine. Leaving it as a macro makes it marginally faster, of course. The remainder of the macros work much like the earlier versions.

So, there we have two approaches to the abstract-data-type stack problem. I developed these routines for doing compiler work, and I've found them to be pretty useful there. Stacks are generally handy anywhere that you need to simulate recursion (that's what's going on in most compiler applications). If you think about it, it's the runtime stack that makes recursion work—every recursive instance of a subroutine pushes a new stack frame (a block of memory that it uses for its own local variables) onto the runtime stack. These variables are then popped when the subroutine returns. You can eliminate recursion by doing this pushing and popping yourself. Instead of calling a subroutine recursively, you can push the local variables onto an explicit stack and then loop. For example, the following recursive subroutine figures n factorial:

```
int fact( int n )
{
     if( n == 1 )
         return 1;
     else
         return n * fact( n-1 );
}
```

In fact, you can do the same thing with a local stack with:

```
int fact_s( int n )
{   int fact;
    stack_dcl  ( stack, int, 20 );
    stack_init ( stack );
    for(; n  1; --n )
        push( stack, n );
    for( fact = 1; !stack_empty(stack) ;)
        fact *= pop(stack);
    stack_clear( stack );

    return fact;
}
```


C Programmers

Explore a world of DOS memory

Beyond 640K

Use PowerSTOR to exploit:

- LIM 4.0 or LIM 3.2 Expanded Memory
- AT Style Extended Memory
- Hard Disk Space
- Any Combination

- PowerSTOR is the easy way to use system memory resources. It self-configures at run time, under control of your software.
- PowerSTOR makes system memory resources available to your software as STORheaps. STORheaps can be allocated and de-allocated dynamically. Up to a gigabyte of PowerSTOR memory can be used as one huge STORheap or many smaller STORheaps.
- PowerSTOR uses only 15K in the lower 640K.
- PowerSTOR with K&R standard source code – only $92.50.

Call Toll Free in the US and Canada

1-800-621-5663

Acme Software

P. O. Box 40734, Portland, Oregon 97240 — Phone (503) 235-2816

Circle 305 on Reader Service Card

VIDEO TAPE

Over 6 hours! This complete C course gives all the in-depth details of ANSI C. Over 200 screens of examples explain syntax, pointers, structures – literally everything! Learn techniques used by expert C programmers. Companion book included. Unconditional 100% satisfaction guaranteed. Order now. Call the 800 number. $74.95 postage paid. No extra charge for COD.

Automation Education

2309 Royce Drive

Arlington, TX 76016

(800) 762-5515

In Canada, add $5.00 & phone number. Overseas, add $30.00. In Texas, add sales tax.

UNDERSTANDING C

Circle 207 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

4c™ Hypertext Source Code Databank for Programmers

Explore source code! Navigate uncharted logic trails easily!

4 c includes:

★ **A Source Code Analyzer:**
The analyzer maps and cross-references functions and variables to their respective path and file names. The cross-reference table can be displayed alphabetically and by type of variable. It allows a programmer to edit functions directly without specifying a file name.

★ **A Point-and-Shoot Zoom Key:**
Which instantly locates any function, global variable, structure definition, #define statement, or macro, etc., in any file, directory, and drive, and displays it in a new edit window.

★ **A Full Featured Built in Editor:**
Featuring multiple files, multiple windows, cut-and-paste between files, search-and-replace, auto indent, tab support, and a familiar user interface with pull-down menus, or use your own editor!

"Given a procedure's identifier, an editor should be able to bring up the full procedure interface, including parameters, so that you don't have to remember how many and in what order. Wouldn't it be nice to hit a hot key on the name of a complicated data type and have its definition pop up in a window? It may seem like a real small matter, but how many of you still flip through printouts doing things the editor should be doing?"

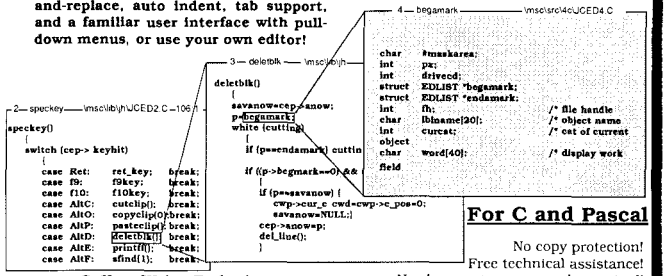
- Phillipe Kahn, "The Connection", Spring/Summer 1988

4 C does this and more

"4c's Magic is apparent..."
- PC Magazine, February 1989

"Daylight in the C Confusion"
- Toolbox magazine, December 1988

"...Destined to become the standard editor for C..."
- Computer Personlich magazine, October 1988



For C and Pascal

No copy protection!
Free technical assistance!
No changes to source code required!
Required IBM PC compatible with 350K RAM.

30 Day Money Back Guarantee!
Only $119.00 plus shipping
Visa, Mastercard, check, money order, or COD.

Call or Write Today!

Tri-Technology Systems Inc.
1225 S. Elgin, Forest Park, Il. 60130
Phone: 312-366-7595

Circle **276** on Reader Service Card

APPLIED INTUITION

ANNOUNCES

WINDOWS ++ VER 2.0

The Complete C++ User Interface Facility.

WINDOWING
- Virtual Screens
- Scroll & Move
- Dynamic Resizing
- Sub "NESTED" Windows
- Menus Bound to Windows
- Write Windows to Disk
- Write Virtual Screens to Disk

MENUS
- Pop Up
- Pull Down
- Spread Sheet
- Menu Compiler
- Context Sensitive Help

DATA ENTRY
- Form Design
- Date & Time
- Social Security #
- Telephone Numbers
- Currency
- Scientific Notation
- Complex Numbers
- Text with Editing
- Contex Sensitive Help
- Automatic Validation

SYSTEM SUPPORT
- PC/MSDOS
- VGA 29/43/50 Line Modes
- EGA 43 Line Mode
- CGA, MDA
- Unix, Xenix
- Curses, Terminfo, Termcap
- Mice
- C++ ver 1.2 or Greater

APPLIED INTUITION

5485 Beltline Rd. Ste. 225
Dallas, Texas 75240
CALL (214) 458-7134

Circle **41** on Reader Service Card

"I recommend getting both (Volumes)... compared to UNIX tools, the Toolbox tools have more options and provide more detailed information..." *---Kenji Hino, The C User's Journal, February 1990*

The C Programmer's Toolbox contains more than 20 tools for the serious C programmer. The tools are available in three different development environments: PC/MS DOS; Macintosh's MPW; and Sun's UNIX. Some of the major tools include:

CDecl	Translates C declaration statements to/from English *(helps you decipher those complicated C variable declarations and construct new ones). {MPW & Sun only}*
CFlow™	Determines how a program is organized and what is in the runtime library *(quickly shows you function hierarchy and interdependancies)*
CHilite™	Highlights and prints your C source code *(great for creating your final documentation). {MPW only}*
CLint™	Syntactically checks one or more of your C source code files *(use this to find those hard to find, syntactically correct programming mistakes)*
Cpp	Preprocesses C source code files that have ANSI macro definitions *(helps you port ANSI C code to/from your target environment). {MPW & Sun only}*
CPrint™	Reformats/beautifies your C source code *(makes code easy to read and ends disputes on source code formats).*
CritPath™	Determine a program's critical path. *(helps you improve your program's performance) {PC DOS only}*
CXref™	Cross references and checks usage of up to 17 different classes of symbols in your C source code *(cross references any size program and number of listings)*
PMon™	Monitors a program's execution *(non-invasively monitors your program's function execution, system service and BIOS usage). {PC DOS only}*

"... The tools I tried operated both quickly and flawlessly..." *---M. Steven Baker, Programmer's Journal, May 1988*

C™ Programmer's Toolbox / MPW PC Sun

The tools include support for ANSI C, Microsoft C, Turbo C, Quick C, Sun UNIX C, Macintosh's MPW C, THINK C and more. The Toolbox easily handles any size program through its virtual memory system. Several tools also include a C macro processor that accepts K&R, ANSI and other C dialects.

C Programmer's Toolbox™
MPW ($295), PC Volume I or II ($99.95),
PC Volume I & II ($175), Sun ($495)

C Programmer's McTool Series™ (Macintosh - non-MPW)
McCPrint™ ($59.95), McCLint ($99.95)

"... it's a complete set of utility programs which can make the life of a C programmer easier... I really like this package..." *-- Scott Robert Ladd, Micro Cornucopia, November 1988*

Call and Order today...
(CA residents: please add sales tax; Overseas: contact us for details)

MasterCard® VISA®

MMC AD Systems™
Box 360845
Milpitas, CA 95036
Phone: (415) 770-0858
Fax: (415) 770-0116

Circle **94** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Best Lint Just Got Better

PC-lint is a C source code analysis tool. It will check code more thoroughly than a C compiler and, because it looks across multiple modules, it enjoys a perspective that your compiler doesn't have.

Control your development time and cost. Find C source code errors before they find you.

Over **270 error messages**. More than **90 options** for complete customization. Suppress error messages, locally or globally, by symbol name, by message number, by filename, etc. **Easy to Use** - If you know C you already know how to use PC-lint. Check for **portability** problems. Alter size of scalars. Adjust format of error messages. Automatically generate ANSI **prototypes** for your K&R functions.

New Features in Version 4.0

- **Lint Object Modules** - incremental linting - big linting.
- **Weak Definial Checking** - macros, structs, enums, typedef's and declarations not used globally and locally. Header files not used. Externals that can be made static. Declarations that can be moved from headers. etc.
- **Library Headers** - avoid redundant prototype processing.
- **Over 70 new messages and 30 new options,** including identifier truncation clashes.
- **New "Elective Notes" message category.**
- **Unix lint compatibility**

Mainframe and Mini Programmers

Get **FlexeLint 4.0** (formerly Generic Lint) in shrouded source form for VAX/VMS, Unix, IBM VM/MVS, QNX OS-9, DG AOS/VS, etc. Requires only K&R C to compile but supports ANSI. Pricing starts at $798. Call for details.

Gimpel Software

3207 Hogarth Lane Collegeville PA 19426

(215) 584-4261 - CALL TODAY

(215)584-4266 - FAX

PC-lint is only $139 - Specify MS-DOS or OS/2

30 Day Money-back Guarantee

Quantity and educational discounts available. PA add 6% sales tax. Outside USA add $20.

PC-lint, FlexeLint and Generic Lint are trademarks of Gimpel Software.

Contact this advertiser directly.

The first 'for' loop is the equivalent of spinning down the recursive calls. Every push mimics what happens on the recursive runtime stack. The second 'for' loop simulates the returns. The stack version is a little larger than the recursive one, but doesn't have the recursive approach problems, such as stack overflow. Of course, it's brain damaged (that's a technical term) to use either recursion or stacks to compute a factorial.

A more practical problem is binary-tree traversal. The main problem here is recursion depth. In a worst-case tree, you'll need as many recursion levels as there are nodes in the tree, and you'll run out of stack in no time at all. If you use a nonrecursive, stack-based traversal method, you have a lot more control over stack depth, and it will be much easier to detect overflow. The amount of memory used for the stack is also smaller because the compiler puts lots of stuff other than the local variables on the runtime stack when it processes a subroutine call. A recursive inorder binary-tree traversal algorithm looks like this:

```
typedef struct tnode
{   int key;
    struct tnode *left;
    struct tnode *right;
} tnode;
traverse( tnode *root )
{
    if( !root )
        return;
    traverse( root-left );
    printf("%d\n", root-key );
    traverse( root-right );
}
```

The following inorder traversal routine uses an explicit stack to keep track of the whereabouts of the the parent node.

```
inorder( tnode *root )
{   stack_dcl  ( stack, tnode*, 20 );
    stack_init ( stack );
    do
    {
        while( root )
        {
            push( stack, root );
            root = root-left;
        }
        if( !stack_empty( stack ) )
        {
            root = pop( stack );
            printf("%d\n", root-key );
            root = root-right;
        }
    } while( !(stack_empty(stack) && !root) );
    stack_clear ( stack );
}
```

Again, the stack-based routine is a little larger than the recursive one, but it's more manageable.

IN CONCLUSION

The techniques used to implement the stack macros are, of course, applicable to other abstract types as well. For example, I have a set of set-manipulation routines that are implemented largely in macros. There are difficulties, too, with the initialization problem a case in point.

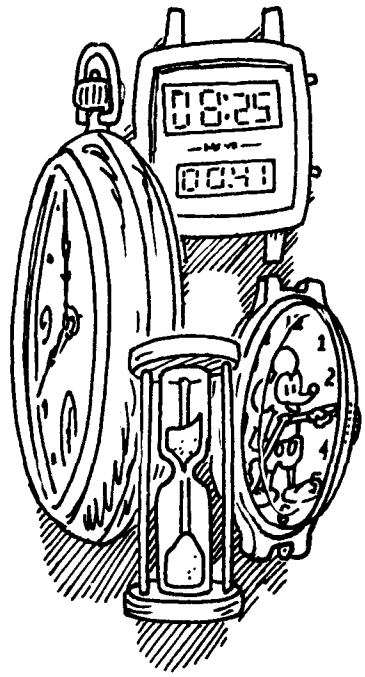
Ideally, the stack would be initialized when you declared it and the associated memory freed when the stack went out of scope. There's no way to do this automatically in C, though you can can use a C++ class for this purpose—I'll demonstrate how next time. PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

By Murray Lesser

DIFFERENT TIMES

Compensating for system clock idiosyncracies



This second article on computers and time presents a system utility that depends on the fact that there is no hardware connection between the DOS system clock and any installed real-time clock. CLKJUST.C resets the system clock from the real-time clock with an elapsed time offset, to correct for the observed error rate of your real-time clock.

This utility will not run correctly under DOS 3.3 or later unless your boot copy of DOS has been patched to remove the ill-conceived software connection between the two clock systems, as described in the previous installment, "Computing Time in C" (*PJ* 8.1). CLKJUST uses the assembled private library modules shown in that article for programmed access to the real-time clock.

In this article, I use the conventions established in Borland's Turbo C, version 2.0. If you are using a different compiler, you will have to translate to its function dialect. In spite of the fact that C has long been the best language for writing system utilities, low-level library functions necessary to manipulate BIOS and DOS objects are not standardized. Most DOS C compilers supply the low-level library functions, but since neither the function names nor associated data structures are part of any standard, system utilities are not portable between compilers.

DOS CLOCK FUNCTIONS

The three assembled modules shown here contain library functions primarily related to the DOS clock. They are useful additions to the DOS-based time functions in your C library, even if you are running on a system that does not contain a real-time clock. As written, they can be linked to Turbo C programs compiled for either tiny or small memory models. I carry the corresponding object modules in my private library.

The **hundsec** function (Listing 1) returns (as a long) the elapsed time since midnight in hundredths of a second, as obtained from the argument time structure. The function is compiler-dependent: the time structure used here is defined in the Turbo C DOS.H header file. The function has been written to use time structure data obtained from either the DOS clock or from the real-time clock.

Using **hundsec** on two readings of the DOS clock allows the time interval between two programmed events to be calculated to the precision of the DOS clock without using floating-point arithmetic. However, time intervals measured from the system clock are not accurate to that precision. The reading produced by any system clock running on a PC will be that corresponding to the most recent *tick* count accumulated in the BIOS timer counter. Most programmers assume that the PC timer clock is running at its nominal frequency, and any reading from the system clock is correct to within 55 milliseconds. Unfortunately, this ideal state isn't very likely. The accuracy with which the timer counter is incremented, and hence the accuracy of the system clock, depends on the actual frequency of the BIOS clock timer oscillator, not on the nominal frequency given in the programming reference manuals.

The specified tolerance on the actual frequency of the PC I/O channel oscillator is plus or minus 0.01% of nominal, according to the *IBM Personal System/2 Hardware Interface Technical Reference*. Since the BIOS clock timer frequency is derived from the high-speed oscillator, its tolerance is no better. While an allowable frequency error of 0.01% may be good enough for a high-frequency computer oscillator, it isn't a tight-enough spec to make a very good time keeper; it corresponds to a maximum allowable timing error of about 8.5 seconds per day. So, although the actual error rate of your PC clock is probably smaller than the maximum allowed, you can't count on elapsed time measurements to be accurate to the nominal timer tick rate if the time interval is greater than about 250 seconds.

Murray L. Lesser is a retired IBM engineer. Murray has written three books dealing with enhancing various Microsoft BASIC compilers with assembled modules. The most recent of these, Advanced QuickBASIC 4.0, *was published by Bantam.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Confusing Code?

```
#include <stdio.h>
text_count(){int c,nlines,nwords,nchars,inword;
inword=NO;nlines=nwords=nchars=0;while((c
=getchar())!=EOF){++nchars;if (c=='\n')
++nlines;if ((c==' ')||(c=='\n'))inword=NO;else
if(inword==NO){inword=YES;++nwords;}}printf(
"%d %d %d\n",nlines,nwords,nchars);}
```

C It Your Way!

```
#include <stdio.h>

text_count ()
{
     int c, nlines, nwords, nchars, inword;

     inword = NO;
     nlines = nwords = nchars = 0;
     while ((c = getchar()) != EOF) {
          ++nchars;
          if (c == '\n')
               ++nlines;
          if ((c == ' ') || (c == '\n'))
               inword = NO;
          else if (inword == NO) {
               inword = YES;
               ++nwords;
          }
     }
     printf ("%d %d %d\n", nlines, nwords, nchars);
}
```

NOW with **C-Clearly™**, format C source code exactly the way you want it. C-Clearly's context sensitive analysis will format any C program in your own personal or corporate style.

EASY to use, C-Clearly's style templates are a snap to modify, since they resemble C source code you edit into your preferred format. Templates are included for several common styles as well as standard K & R.

LISTINGS can also be created with function names and comments highlighted for improved readability. Listing options include line numbers, headers and/or footers and flow lines.

IDEAL for making obtuse code clear. Allows all of your source code to be presented in a consistent format of your choosing. Also great for code walkthroughs and final documentation listings.

WORKS with all IBM PC, XT, AT, PS/2 and compatibles, with 512K RAM. Automatically processes all include files and pre-processor statements. ANSI-C compatible. Not copy protected.

C-Clearly **$129.95**

Shipping & Handling USA $5; Canada/Mexico $10; Other Countries $15; CA Residents add sales tax. Visa/MasterCard/COD accepted.

For orders or information call:

1-800-662-8266

V COMMUNICATIONS, INC.
4320 Stevens Creek Blvd., Ste 275, Dept. PJ6
San Jose, CA 95129 (408) 296-4224

Circle 341 on Reader Service Card

The function **newtime** is not compiler-dependent. It resets the PC time counter (and hence the DOS clock time) by converting a long hundred-seconds value (such as returned by **hundsec**) to the equivalent tick count. The conversion is based on the clever algorithm used in the PC-DOS CLOCK$ driver, allowing a 32-bit result to be produced from 16-bit arithmetic. I have modified that algorithm by half-adjusting the quotient from the final division rather than truncating it, thus avoiding DOS's "lost tick" problem.

Together, these two clock functions permit the DOS clock setting to be adjusted by increments to the nominal precision of 0.01 second, using only long-integer arithmetic.

The third function, **days**, calculates the number of days that have elapsed since January 1, 1980, the epoch of the DOS calendar. January 1, 1980 was a Tuesday, so you can calculate a day-of-week code (where 0 = Sunday) using C's **remainder** operator [%]. Add two to the return from **days** and then calculate the remainder after dividing the sum by seven.

The **days** function is not compiler-dependent, as I have used integer values of the year, month, and day components of the input date, rather than any compiler-dependent date structure. This function is used primarily to calculate the difference in days between two events, one of which may be marked by the creation date of a file. Determining elapsed calendar time in days from the **days** function is computable for all valid input dates; determining elapsed time in seconds from the ANSI C standard **difftime** function is not.

The only clock known to keep exact time over a very long period is the cesium atomic clock, the basis for the time signals broadcast by the Bureau of Standards and other national time-signal organizations. Even "good" quartz clocks are subject to some error. The effect of the average difference between a clock's actual oscillator frequency and its nominal value, measured over a reasonable period, is called the clock rate. Rate is usually measured in tenths of a second per day, gaining or losing. However, the rate of even the best quartz clock will vary somewhat over the long term, due to variations in temperature and battery voltage and to component aging.

I have never measured the rate of the BIOS clock in any of my systems because I never leave them on for days at a time. I have measured the rates (averaged over about a 15-day period) of the real-time clocks in two of my machines: the one in my PS/2 model 80 gains about 0.2 second per day, and the one in my PC Convertible loses about 0.8 second per day.

CORRECTING THE CLOCK

If you are not happy with the average rate of your clock, there are three ways to make it keep better time. One is to reset it often to a time standard. Cheap quartz clocks (tending to have rates of several seconds per day) make this easy; they provide an "adjust" button to reset the displayed time to the nearest minute.

The second method is to regulate your clock. Some quartz clocks have a trimmer screw for the frequency oscillator. However, there doesn't seem to be any way to mechanically regulate the real-time clocks in the PCs.

The third method is to correct the clock reading for the accumulated error since the clock was last reset, using the measured value of the rate. If the rate is known to the nearest 0.1 second per day, you will have produced an equivalent clock keeping time to within about 1.5 seconds per month—not counting leap seconds.

The system utility CLKJUST.C (Listing 2) was designed to use the third method of correction: it corrects the time shown by the real-time clock for its rate and the number of days since it was last reset, and then uses this value to set the DOS clock. The source code shown is not only system-dependent in the DOS and compiler sense, but must be edited to fit your specific hardware system. As shown, it assumes a perfect clock (RATE = 0) and an arbitrary directory containing the necessary historical data file. The listing is fairly well commented, so I will discuss only some of the underlying philosophy.

The program will run in any of three modes, as shown in the prologue to the listing. Although all times used are displayed to a precision of 0.01 second, you and I know that this is for program-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

mer convenience and has little relationship to reality. I would consider that displayed clock errors of less than about 0.2 second are not significant, especially considering operator reaction time in hitting a key in response to an audible time signal.

The date at which the real-time clock was last reset is stored on disk as a zero-length file (directory entry only) having the filename TIMEFILE.XX, where the XX is the computed setting error of the real-time clock in hundredths of a second. Unless you keep your TIMEFILE in subdirectory C:\TIME, you will have to edit the source code, changing the defined string *TIME_DIR*.

The program won't run unless it can find TIMEFILE. If the program has never been run before, a dummy starting value (no extension) will be written. However, only the Reset mode can use this dummy file, and any other call will terminate with a fatal error.

The other thing you have to tell the program before compilation is the rate of your clock. I started by defining *RATE* as zero (as shown in the listing) and used the program itself to calibrate my real-time clocks over at least ten days before entering a value into the *RATE* definition. (Turbo C will show two "code has no effect" warning messages during compilation when *RATE* is defined to be zero. Ignore them.)

The program operates in three modes. If the calling command line includes an argument that begins with the letter **R** (for Reset), the real time clock is reset from the system clock after calling the built-in DOS commands DATE and TIME. This update is made in the local function **setclocks**. If no new date nor time is given, the real-time clock will be reset from the existing values in the DOS clock (probably losing one tick in the process). Note that reset is inhibited by looping during the last two seconds before midnight, to avoid possible date errors. The real-time clock will be reset at the time given by the first DOS clock reading marking the start of a new second.

After resetting the real-time clock (from any source) an updated TIMEFILE.XX is written by the local function **makefile**. The XX value for TIMEFILE is determined by re-reading the DOS clock at the beginning of the next real-time clock second, thus effectively setting the real-time clock (for this program) to the same precision as used by the DOS clock.

If the calling command line has an argument that begins with the letter **A** (for Adjust), the program merely updates the system clock and terminates. This is the batch file mode. Since it was too much trouble to correct the system clock across midnight, the program terminates with an exit(2) in such a case. Fatal errors terminate with exit(1). Thus, the command **CLKJUST A** in my AUTOEXEC.BAT file is preceded by a *label* and followed by the two commands in sequence: **IF ERRORLEVEL 2 GOTO** *label* and **IF ERRORLEVEL 1 PAUSE**. The first command will recycle the program until the midnight problem has passed; the second attracts attention to a problem requiring manual intervention.

If you keep power on your system 24 hours a day, you can minimize the effects of the PC clock timer tolerance by running CLKJUST in Adjust mode at reasonable intervals.

Turn your favorite C compiler into a powerful database manager with the
C/Database Toolchest
The C/Database Toolchest™ adds sophisticated file management functions to your Power C™, Turbo C®, QuickC®, or Microsoft® C compiler. With the C/Database Toolchest™, your data requires much less disk space than with programs like dBASE®, and you can access your data much faster. Of course the full power of C provides you with an unlimited amount of programming flexibility.
The C/Database Toolchest™ includes three major components:
1) An advanced B+tree library gives you instant access to your data.
2) A high-level ISAM library provides you with an easy-to-use C interface, and
3) A complete database manager (with C source code included) shows you how to create impressive applications.
You also receive a comprehensive 350 page manual and a utility for converting dBASE® files.
The C/Database Toolchest™ supports features that you'd expect to find only in products costing ten times as much. Advanced features include variable length records, variable length keys, multiple keys per index, and multiple indexes stored in a single file. Your data files can contain an unlimited number of records, and each record can be as large as 32K bytes in length.
About the only thing that the C/Database Toolchest™ doesn't do is cost you a lot of money. We've kept our price low so you can manage your budget as easily as your data.
Now Only $19.95!
Order Coupon
Name
Street
City
State Zip
Telephone
Paying By Money Order Check
Visa MC AX Disc.
Card#
Exp. Date
Disk Size 5¼" 3½"
Qty. Product Price Subtotal
C/Database Toolchest $19.95
C/Database Library Source. $10.00
B+tree & ISAM library source code
Add Shipping ($5 USA, $20 Foreign)
Texas Residents Add 8% Sales Tax
Total Amount of Your Order
Order now by calling our toll free number or mail the coupon to:
Mix Software
1132 Commerce Drive
Richardson, TX 75081
1-800-333-0330
60 Day Money Back Guarantee
Not Copy Protected ■ Royalty Free
For technical support, please call:
1-214-783-6001
MIX software
C/Database Toolchest and Power C are trademarks of Mix Software. QuickC and Microsoft C are registered trademarks of Microsoft. Turbo C is a registered trademark of Borland. dBASE is a registered trademark of Ashton-Tate.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
        .model  small
; These functions are linkable to Turbo C small/tiny model programs.
; Assembled with Microsoft MASM v 5.1, switch /MX
.data
; Months table: number of days in (non-leap) year prior to first of
;               month corresponding to entry index:
months  dw      0,31,59,90,120,151,181,212,243,273,304,334

.code
        public  _hundsec,_newtime,_days
; long hundsec(struct time *t) It returns the time of day passed in the
; argument structure as 100 times the number of seconds that have elapsed
; since midnight. The time data structure is in the Turbo C v2.0 dos.h
_hundsec        proc
        push    bp              ;Save significant registers
        mov     bp,sp
        push    si
; Calculate value of seconds * 100 passed in time structure:
        mov     si,4[bp]        ;Address of time structure
        mov     cx,[si]         ;Hours in CH, minutes in CL
        mov     al,60
        mul     ch              ;Hours -> minutes in AX
        xor     ch,ch
        add     ax,cx           ;Hours + minutes -> minutes in AX
        mov     cx,6000
        mul     cx              ; -> seconds * 100 in DX:AX
        mov     cx,ax           ; low order in CX
        mov     bx,2[si]        ;Seconds, hundredths in BX
        mov     al,100
        mul     bh              ;Seconds * 100 in AX
        xor     bh,bh
        add     ax,bx           ; + hundredths
        add     ax,cx
        adc     dx,0            ;Total time -> seconds*100 in DX:AX
        pop     si
        pop     bp
        ret
_hundsec        endp

; void newtime(long s)   This resets the system clock to the time
; represented by s, one hundred times the number of seconds that have
; elapsed since midnight. This routine is based on the  algorithm used in
; the PC-DOS 3.3 CLOCK$ driver.
_newtime        proc
        push    bp
        mov     bp,sp
        mov     ax,4[bp]        ;Low word of seconds * 100
        mov     bx,59659
        mul     bx              ;Low word multiplication
        mov     cx,dx           ;Keep top word of result in CX
        mov     ax,6[bp]        ;High word of seconds * 100
        mul     bx              ;High order result in DX:AX
        add     ax,cx
        adc     dx,0            ;Count * 5 (shifted) in DX:AX
        xchg    ax,dx           ;  Now in Ax:DX
        mov     bx,5
        div     bl              ;High "word" of count in AL
        mov     cl,al
        xor     ch,ch           ;Now in CX where it belongs
        mov     al,ah           ;Put remainder of first division in AL
        cbw
        xchg    ax,dx           ;Total "remainder" now in DX:AX
        div     bx              ;Finish division
        cmp     dx,2            ;Check for remainder >= half divisor
        mov     dx,ax           ;Low order of count in DX
        jle     pass            ;Skip half-adjust if not needed
        inc     dx              ;Half-adjust
        adc     cx,0            ;Propagate any carry in DX
; Load result into the BIOS data timer count
pass:   mov     ah,1
        int     1ah
        pop     bp
        ret
_newtime        endp

; unsigned days(int year, int month, int day)  This returns the number of
; elapsed since January 1, 1980.
year    equ     4[bp]           ;Offset of year argument in stack
month   equ     6[bp]           ;          month argument
day     equ     8[bp]           ;          day argument
_days   proc                    ;Calculate days since 1-1-80
        push    bp
        mov     bp,sp
        mov     ax,year
        sub     ax,1980         ;Years since 1980
        mov     bl,4
        div     bl              ;Quot in AL, rem in AH
        mov     cl,ah           ;Save remainder
        cbw                     ;Clear AH
        mov     bx,1461         ;Days in four years (365*4+1)
        mul     bx              ;Cumulated days in AX, DX = 0
        push    ax              ;Save count to date
        mov     al,cl           ;Years since last leap year
        cbw
        mov     cx,365
```


Complete Modem Communications

Using Sugar Coated Serial

Now for both **C** and **Turbo Pascal** Programmers

Features:

- Complete pull-down menu communications program and its commented source code using only SCS routines.
- Supports XModem, YModem, ASCII, & Telink protocols.
- All source code included including assembler, with many predefined constants that take the pain out of understanding and modifying nitty-gritty, low-level aspects of communications.
- No knowlege of assembly language is required...procedure calls are as simple as "MakeCall(PhNumStr)" and "SendFilesYModem.
- Versions for TP v5.0 or greater and Microsoft & Turbo C.
- Complete user manual including an informative and practical discussion on the theory of serial communications.
- No royalties on programs you sell
- Interrupt driven with no TSR requirement.
- Pull-down menu source library included.

Complete system	$109.95	(For Turbo Pascal or C)
Demo Disk	$5.95	($6.95 for 3½" disks)
Information pak	Free	

Shipping included, IL residents add 6.25% sales tax. Please specify disk size and language (C or Turbo Pascal.)

For credit card orders or info call **(708) 759-1270** or send check or money order to:

Goin Places Software
Suite 145
1202 N. West 75th Street
Downers Grove, IL 60516

Visa MC

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        mul     cx                  ;Additional days to first of year
        mov     bx,month
        mov     cx,day
        or      ax,ax               ;If zero, current year is leap year
        jnz     ok                  ;Skip correction if not leap year
        cmp     bx,2                ;If first two months in
        jg      ok                  ;  leap year, correct day
        dec     cx                  ;  counter to adjust
ok:     dec     bx                  ;Set up month index in BX
        shl     bx,1
        pop     dx                  ;Days in previous 4-year blocks
        add     ax,dx               ;Days to first of current year
        add     ax,months[bx]       ;Days to first of current month
        add     ax,cx               ;Total days since 1-1-80
        pop     bp
        ret
_days   endp
        end
```

LISTING 2

```
/* CLKJUST.COM uses the observed rate of the real-time clock and a file
(<path>TIMEFILE.xx) written at the time the real-time clock was last
set, to "correct" the system clock time accordingly. Call with:
  CLKJUST A - to set the system clock to the corrected time and date (batch)
  CLKJUST R - to reset both the system and real-time clocks to a new time
  CLKJUST - to display the corrected time continuously (calibrate mode).
When in calibrate mode, the display will be stopped when any character
key is pressed. If the ENTER <CR> key is pressed instead, both clocks
will be reset to the nearest 30 seconds. A new TIMEFILE will be written
whenever the real-time clock is reset. TIMEFILE.xx is a zero-length file
(directory entry only) written to the directory given by <path>, where
xx is the string representation of the fraction of a second (in hundredths)
the real-time clock was "fast" with respect to the system clock at the
time it was reset. CLKJUST returns an exit code to DOS on completion.
  0 - indicates successful completion
  1 - indicates a fatal error, manual intervention required to correct
      the fault indicated in the exit error message.
  2 - indicates if the program attempted to "correct" the sytem clock
      across midnight (in either direction). This condition can be
      corrected by rerunning the program until sufficient time has passed.
Compiled using Turbo C v2.0 with -mt -lt -O options.
WARNING:  This program will not operate correctly when running under a
  version of DOS that automatically resets the real-time clock when the
  DOS clock is reset.  If running under DOS 3.3 (or later), the DOS
  system must be patched to eliminate this undesirable idiosyncracy.
*/
#define TIME_DIR "C:\\TIME\\"   /* Directory with TIMEFILE.xx */
#define RATE 0  /* Real-time clock rate: hundredths sec/day; gain is plus */
#include <conio.h>      /* PC-specific kbd/display functions */
#include <ctype.h>      /* Toupper() function prototype and macro */
#include <dir.h>        /* Directory function prototypes */
#include <dos.h>        /* MS-DOS structures and function prototypes */
#include <fcntl.h>      /* Low-level file "open" flag definitions */
#include <io.h>         /* Low-level file function prototypes */
#include <stdlib.h>
#include <string.h>

int    elapsed, delta, handle;
unsigned todayday;      /* Elapsed days since January 1, 1980 */
long   seconds;         /* 100 times elapsed seconds since midnight */
char   *p, filename[63];
struct time now, clock;
struct date today;
struct ftime then;              /* File directory time and date */
struct ffblk fblock;            /* Expanded file directory data */
int setclocks(void);           /* Local functions in this module */
int makefile(void);
char *months[13] =  { "","January","February","March","April","May","June",
           "July","August","September","October","November","December" };
```

continued

If CLKJUST isn't given a command-line argument, the newly corrected DOS clock time will be displayed continuously, in place, until a key is pressed. Note that if you are running this program on a 4.77 MHz machine, the standard **printf** function is not fast enough to show you every clock tick. However, Turbo C's **cprintf** writes directly to the display screen memory, and will show every tick on a slow system.

Pressing ENTER will reset the real-time clock to the nearest 30-second value. I used the 30-second time (rather than the even minute) for ease of programming; there is no requirement ever to reset the minutes, hours or date values. Also, all national time-signal stations leave out the 29th tick in every minute, making the 30-second tick easy to distinguish.

A CAVEAT

Running a program that resets the system clock in a multitasking environment, such as under DESQview, may have a deleterious effect on other programs executing simultaneously. Thus, it is safest to run CLKJUST directly under DOS. PJ

The Heap Expander™ v 2.05

Gain virtually unlimited heap space.

Function
Provides interface routines so that your programs use expanded memory, extended memory or conventional memory. As memory becomes full, The Heap Expander swaps to disk.

Utility
The Heap Expander comes with complete source code, libraries and documentation plus four utility programs that simplify development of expanded memory programs.

Guarantee
Guaranteed by The Tool Makers to perform as specified with an unlimited money-back refund.

Includes libraries and source code for all current versions of Turbo C and Microsoft C.
System requirements: IBM PC, XT, AT or clone compatible with MS-DOS or PC-DOS v. 2.0 or above.

MC/VISA/COD
800-248-1045 ext 100 USA
800-952-5560 ext 100 Idaho

still just **$79.95**

THE TOOL MAKERS
P.O. Box 2151
Santa Cruz, CA 95063
408-458-0690

Circle 166 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
char *weekday[7] = { "Sunday","Monday","Tuesday","Wednesday",
                     "Thursday","Friday","Saturday" };

void main(int argc, char *argv[ ])
{
/* Check for functioning real-time clock: */
   if (clktime(&now) != 0)      /* Real-time clock inoperative */
   {
      cputs("\n\n\rWARNING:  Real-time clock not operating."
      cputs("\n\r          Terminating program");
      exit(1);
   }
/* Find (or create) current version of TIMEFILE: */
   strcpy(filename,TIME_DIR);
   if (findfirst(TIME_DIR "TIMEFILE.*",&fblock,0) == 0)
   {                                          /* If TIMEFILE exists */
      strcat(filename,fblock.ff_name);        /* Get full path/name */
      if (strchr(filename,'.') == NULL)       /* If has no <ext>    */
         strcat(filename,".");                /*   add dummy        */
   }
   else
   {
      strcat(filename,"TIMEFILE.");           /*  Create dummy      */
      handle = _creat(filename,0);            /*  file for later    */
      close(handle);                          /*  deletion test.    */
   }
   p = strchr(filename,'.') + 1;              /* Start of <ext> */
   if (argc > 1)
   {
      if (toupper(argv[1][0]) == 'R')
         return(setclocks());                 /* Set both clocks */
      else
      {
         if (toupper(argv[1][0]) != 'A')      /*"A" adjusts system clock*/
         {                                    /* Terminate with message*/
            cputs("\n\n\rUnkown command-line argument.  Use:\n");
            cputs("\n\r   \"CLKJUST R\" to reset both clocks.");
            cputs("\n\r   \"CLKJUST A\" to adjust system clock.");
            cputs("\n\r   \"CLKJUST\"   to calibrate real-time clock.\n\r");
            exit(1);
         }
      }
   }
/*  Adjust system clock to "corrected" real-time clock values.  Start by
    getting previous "delta" (offset between real-time clock and system
    clock when real-time clock was reset). */
   if (p[0] == '\0')                /* No <ext> on TIMEFILE */
   {
      cputs("\n\r     "TIME_DIR "TIMEFILE.* invalid.");
      cputs("\n\r     Run \"CLKJUST R\" to reset both clocks.\n\r");
      exit(1);
   }
   delta = atoi(p);
/*  Get elapsed time in days between time TIMEFILE was written and today: */
   handle = _open(filename,O_RDONLY);
   getftime(handle,&then);                  /* Date/time of file creation */
   _close(handle);
again: /* Repeat point if clock passes midnight between */
       /* reading date & time */
   clkdate(&today);                         /* Real-time clock date  */
   todayday = days(today.da_year,today.da_mon,today.da_day);
   elapsed = todayday - days(then.ft_year+1980,then.ft_month,then.ft_day);
   clktime(&now);              /* Reset system timer and display output: */
   if ((now.ti_hour == 0) && (now.ti_min == 0) && (now.ti_sec == 0))
      goto again;                  /* Real-time clock passed midnight */
   seconds = hundsec(&now) - delta - RATE * elapsed;
   if ((seconds > 8639950L) || (seconds < 0L))
   {
      cputs("\n\r System clock correction would cross midnight");
      cputs("\n\r Rerun program\n");
      exit(2);                     /* Error correctable by rerun */
   }
   newtime(seconds);   gettime(&now);   setdate(&today);
/* Display results: */
   cprintf("\n\r\n   Today is %s, %s %d, %d\n\r\n",
        weekday[ (todayday + 2) % 7],
        months[today.da_mon],today.da_day,today.da_year);
   cprintf("Real-time clock was set 0.%02d seconds fast %d days ago.\n\r",
           delta,elapsed);
   cprintf("Additional Rate correction is %+d hundredths of a second\n\r",
           -RATE * elapsed);
   if (argc > 1)                            /* "Adjust" command line */
   {
      cprintf("\n   System time corrected at %02d:%02d:%02d.%02d\n\r\n",
               now.ti_hour,now.ti_min,now.ti_sec,now.ti_hund);
      exit(0);                  /* Normal exit */
   }
/* Following routine runs only if in "Calibrate" mode: */
   cputs("\n\rPress any character key to stop display.");
   cputs("\n\rPress <ENTER> to reset real-time clock to nearest");
   cputs("30 seconds");
   cputs("\n\n\rCurrent corrected system clock time:\n\n\r");
   while (kbhit() == 0)
   {
      gettime(&now);
      cprintf("     %02d:%02d:%02d.%02d\r",
              now.ti_hour,now.ti_min,now.ti_sec,now.ti_hund);
   }
   if (getch() != '\r')
      exit(0);                  /* Normal exit */
/* Reset both clocks to the nearest 30 seconds: */
   now.ti_sec = 30;     now.ti_hund = 0;
   newtime(hundsec(&now));      /* Reset system timer */
   timeset(&now);               /* Reset real-time clock */
   return(makefile());          /* Make new TIMEFILE.* */
}

int setclocks()  /* Uses DOS date and time functions to reset system clock,
                    then resets real-time clock at "even second" on system clock */
{
   clrscr();
   cputs("\n\n\rWARNING: This routine will reset both system and ");
   cputs(" real-time clocks.");
   cputs("\n\r    Press \"Y\" key if you wish to continue:  ");
   if (toupper(getche()) != 'Y')
   {
      cputs("\n\n\r   Program terminated by user\n\r");
      return (1);
   }
/* Use built-in DOS functions to reset system clock: */
   cputs("\n\n\r");    system ("date");
   cputs("\n\r");      system ("time");
/* NOTE: There is a potential "date" error if the real-time clock is
   reset at midnight.  Thus, if we get here during the last two
   seconds before midnight, the program waits until midnight. */
   gettime(&now);
   while ((now.ti_hour == 23) && (now.ti_min == 59)
               && (now.ti_sec >= 58))
      gettime(&now);
/* Now wait here for the first tick that puts the system clock into the
   following second: */
   while (now.ti_hund/10 == 0)  /* Don't use an initial zero value */
      gettime(&now);
   while (now.ti_hund/10 != 0)  /* Wait for first zero value after */
      gettime(&now);                    /*   a series of non-zeros   */
   timeset(&now);
   return (makefile());
}

int makefile()          /* Set real-time clock date and build new */
{                       /* TIMEFILE file with current delta     */
   getdate(&today);      dateset(&today);
   cprintf("\n\r  Real-time clock reset at %02d:%02d:%02d on %s %d, %d",
          now.ti_hour,now.ti_min,now.ti_sec,
          months[today.da_mon],today.da_day,today.da_year);
/* Real-time clock can be up to one second fast relative to time used to set
   it. Calculate current value of setting "error" and write corresponding
   new TIMEFILE: */
   if (unlink(filename) != 0)   /* Cannot delete old filename */
   {
      cputs("\n\n\rWARNING:  Error in deleting old TIMEFILE file");
      cputs("\n\r          Real-time clock reset invalid\n\r");
      return (1);
   }
   clktime(&now);               /* Real-time clock time on second */
   gettime(&clock);             /* Corresponding system-clock time */
   if (now.ti_sec == clock.ti_sec)
      delta = 0;                /* Real time clock may be 1 tick slow */
   else
   {
      delta = 100 - clock.ti_hund;
      if (delta == 100)         /* Exactly 1 second fast! */
         delta = 99;            /*  reset to avoid overflow */
   }
   itoa(delta,p,10);            /* Value of delta as filename <ext> */
   cprintf("\n\r          %d hundredths of a second fast\n\r",delta);
   handle = _creat(filename,0);
   close(handle);
   return(0);
}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Image

Database Processing
Storage Management

Image is an innovative C library with an unprecedented combination of relational power, efficient memory management, and flexible C-language control.

You gain power! *Image* operations treat entire data sets as units. Sort, search, select or update any number of records with a single call.

You retain control! *Image* always allows direct access to your data. Any C operation, any record, any time.

You get started quickly! *Image* is easy to use, with plenty of sample programs to help you get started.

You finish sooner! *Image* programs usually work the first time. *Image* gives database power with elegant simplicity. Complex problems become simpler.

Your code is efficient! *Image* gives you an integrated family of fast, tight, byte-oriented operations.

You save time and space! *Image* keeps sets in expanded memory or RAM, spilling to disk automatically, and only as needed.

No Royalties — Satisfaction Guaranteed

Venn Set Processing Software
P.O. Box 1008
Ann Arbor, MI 48106

Only $129.95

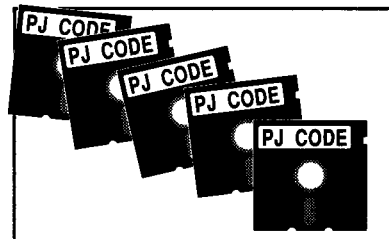
☐ 5.25 ☐ 3.5 Density ____
☐ Microsoft ☐ Turbo ☐ Zortech
☐ Visa ☐ M/C ☐ COD

NAME _____
ADDRESS _____
CITY _____
STATE ____ ZIP _____

_____
Exp _____

ORDER TODAY 1-800-628-8369

Circle **29** on Reader Service Card



On disk from:
Programmer's Journal
P. O. Box 70167
Eugene, OR 97401-0110

Call toll free:
800-234-0386
In Oregon call:
503-747-0800

All Code Disks $10.00 each.
To order, see ad on page 88.

PJ Code DiskScriptions
are available for only $39.95.
To order, see ad on page 8.

On BBS from:

Buffer Board
538 Cardinal Way
Springfield, OR 97477
(Data) 503-747-7636
The Buffer Board contains all of PJ's code from Sept/Oct 1987 issue through the current issue.

EDI
1332 Hermosa Ave. #7
Hermosa Beach, CA 90254
213-374-3478
Now on Tymnet.
Call for your local access number.

Generation Five Inc.
12061 Tech Road
Silver Spring, MD 20904-9998
(Voice) 301-680-3600
(Data) 301-680-9300
Accessible via PC Pursuit, Washington, D.C.
From Files submenu, select "P" for Personal Downloads.

Gilmore Systems of Southern California
5447 Indian Hills Drive
Simi Valley, CA 93063
(Voice) 805-582-1360
(Data) 805-582-9306

Ithaca Street BBS
1145 Ithaca
Boulder, CO 80303
(Voice) 303-494-8865
(Data) 303-494-8868

Mnematics Videotex
P.O. Box 19
Sparkill, NY 10976-0019
(Voice) 800-322-3633
(Voice) 914-365-0184 in N.Y.
Mention *PJ* when you call.

PCS-49 on CompuServe
Section 10
Sponsored by:
MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170
(Voice) 703-450-4181

The Silicon Zone
820 Tivoli Circle #203
Deerfield Beach, FL 33411
(Voice) 305-360-9362
(Data) 305-360-9376

Free Product Information

Use the Reader Service Card (found between pages 88 & 89) to get valuable information on the products and services that appear in *Programmer's Journal*. Just circle the number that corresponds to the number at the bottom of each advertiser.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

By John Otken

Errors and stream I/O

Adapting error and input/output functions from the C library

This column adapts some error processing and stream I/O functions from the standard C library. The error processing routines described here provide a straightforward way for programs to return error information to the user. With the stream I/O functions, programs can read and write multiple files of significant length.

STANDARD OUTPUT

Before discussing error processing or stream I/O, I want to add a couple of standard output routines to the library. The first of these is **putchar**. This is the routine from the C library that writes a character to standard output (stdout). There are several ways an MS-DOS program can write to stdout; one method is as follows:

```
;;      putchar
;
;       entry   AL      character
putchar proc
        pushm   ax,dx
        mov     ah,2
        mov     dl,al
        int     21h
        popm    dx,ax
        ret
putchar endp
```

A faster method uses an undocumented DOS function. INT 29h displays the character in AL. A quick check with a debugger reveals that DOS invokes BIOS INT 10h function 0Eh for each INT 29h.

```
;;      putchar
;
;       entry   AL      character
putchar proc
        int     29h
        ret
putchar endp
```

Since INT 29h neither supports redirection nor handles tab characters, your code should include those. Redirection is difficult to support, but tabs are easy. Send a tab with INT 21h.

Another C output routine is **puts**. This function writes a string and newline to standard output. Standard C does not define a put string routine without the newline, so I wrote my own, which I called **put_string**. Both **puts** and **put_string** have the same entry and exit conditions.

```
;;      put string
;
;       entry   DS:SI   asciiz string
;       uses    AX
;
put_string proc
        push    si
        jmp     put2
put1:   call    putchar
put2:   lodsb
        cmp     al,NULL_CHAR
        jne     put1
        pop     si
        ret
put_string endp
```

Because there are very few functions in the C library that use underscore characters, I can improve the naming convention in my assembly language library. Procedure names without underscores are direct derivatives of the standard C library, while those with underscores are assembly language procedures. For example, **puts** is from the C library; **put_string** is not.

The last new output routine we will look at is **put_unsigned**. This writes an unsigned decimal number from AX to standard output, and is similar in operation to **putw**.

ERROR PROCESSING

The C library has a defined error-reporting mechanism. When an error or exception occurs during a function, the function returns a generic error flag (0, EOF, or NULL). The program can inspect the global location *errno* for a numeric code corresponding to the problem. The program can then try to recover from the error, or it can simply display a message about the problem.

The C library has two functions for error processing. The first, **strerror**, returns error strings. The second function, **perror**, writes error strings to the standard error device (i.e., the console). The *errno*, **strerror**, and **perror** mechanisms are easy to clone in assembly language, but they have shortcomings. The system is difficult to extend to errors outside the library, and there is no way to load only a few error strings. If a program uses a single string, it gets them all.

IMPROVED STRERROR

The improved system shown here loads error strings on demand and permits any module from the library or main program to define errors. It has a simple design and code which will be familiar to C programmers. Rather than translate error codes, this system works directly with sixteen-bit string addresses. The **strerror** function is the basis for this system. The C prototype for **strerror** is:

```
char *strerror(int errnum);
```

Pass **strerror** an error code, and it returns a pointer to an associated error string. I assume one normally calls **strerror** with *errno* as the parameter. If there are no error codes, the assembly

---

John Otken is the president of Soft Advances. In the summer he likes to swim, bike, run, and loiter around the beach. He should have a surfboard by now, so don't bother trying to reach him at 10811 Washington Blvd., Suite 205, Culver City, CA 90232, (213) 559-7015.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

language **strerror** can return a pointer to a "current" error string. In order to avoid confusion, I changed the procedure name to **get_strerror**.

```
;;      get strerror
;
;       exit    DS:SI   error string
;               Zf      if no error string
;
```

To set an error string, call the procedure **set_strerror**. It accepts a nonzero string offset in AX. The string must be in DGROUP. **Set_strerror** returns with the carry flag set, which is a common way to indicate errors in 80x86 programs.

```
;;      set strerror
;
;       entry   AX      DGROUP offset of asciiz
;                       error string
;       exit    Cf      1
;
```

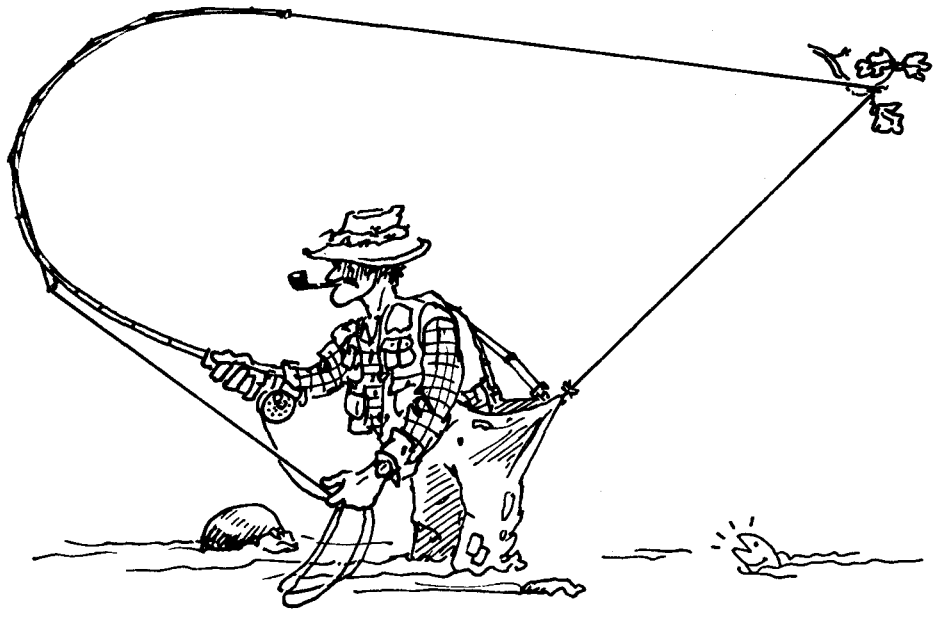
To clear the current error string, call the procedure **clear_strerror**. Direct access to the error pointer is not available. As a result, you must resort to using the procedure interface. This parallels the notion in C that *errno* is not a data object, but rather an lvalue macro.

The next error-handling procedure is **perror**. It works just like the corresponding C function. **Perror** writes a prefix string, the current error string, and a newline to the standard error device. If the prefix string is NULL, **perror** ignores it.

USING THE FUNCTIONS

The following example shows how to use the error functions. The first part has a procedure that divides two numbers. If the divisor is zero, the function sets an error string and returns with the carry flag set.

```
        .data
divide_by_zero  db      'Divide by zero',0
        .code
;;      divide
;
;       entry   AX      dividend
;               DL      divisor
;       exit    AL      quotient
;               AL      remainder
;               Cf      if divisor zero
;
divide  proc
        cmp     dl,0
        je      divl
        div     dl
        clc
        ret
divl:   lea     ax,divide_by_zero
        jmp     set_strerror
divide  endp
```



The second part calls the procedure. If this function detects an error, the procedure outputs an error message. When the program runs, it displays a "Divide by zero" message on the console.

```
main    proc
        mov     ax,123
        mov     dl,0
        call    divide
        jc      mail
        ...
mail:   mov     si,NULL_POINTER
        call    perror
        ...
```

If an error string is common to several independent library modules, put the string in a separate module and reference it as an external. If any of the modules load, the linker also loads the module with the error string.

```
        .data
        extrn   out_of_memory:byte
        .code
        ...
        lea     ax,out_of_memory
        call    set_strerror
        ...
```

An error-handling mechanism based on string pointers is perfect for command-line oriented programs. When an error occurs, display a message and abort the program or command responsible for the error. What is perfect for a simple user interface, however, is inadequate for a complex interface. Consider a failure while opening a file. If the file does not exist, the program should let the user try again. If the program is out of memory, the program should display an error message.

Successful error recovery must test the error condition. In the current scheme, the program must compare an error string pointer with a known string address. This is quite awkward. Two ways to clean up this mess are to have new library functions that test for specific errors (such as "file not found") or that return numeric error codes from certain library functions. Note that the library functions in this column do not currently implement either of these suggestions.

ERRORS IN DOS FUNCTIONS

Errors such as "file not found" are common during MS-DOS functions. The easy way to handle these errors is to have a special DOS procedure that sets the appropriate error string when an error occurs; **ms_dos_strerror** shows how this can be done.

This procedure executes the DOS function and checks for the error flag in the carry bit. If no errors have occurred, the procedure exits. If the error flag is on, **ms_dos_strerror** uses the error number in AX to select an error string. It sets the error string pointer via **set_strerror** and returns to the calling program without changing the registers or carry flag. The calling program can display the error string with **perror**.

Ms_dos_strerror only supports the limited set of error codes MS-DOS returns in AX. A future version should use the extended error information available from DOS function 59h. If

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
; NOTE this listing contains only excerpts from STREAM.ASM
; All of the listings are included on the code disks and at PJ BBS sites

include     asm.inc

SIO_BUFSIZ   equ 8192                       ; stream io buffer size

stream_io_str struc
  sio_position        dd  ?                 ; stream position
  sio_offset          dd  ?                 ; buffer offset
  sio_read_bc         dw  ?                 ; byte count for reading
  sio_write_bc        dw  ?                 ; byte count for writing
  sio_file_handle     dw  ?                 ; dos file handle
  sio_buffer          db  SIO_BUFSIZ dup(?) ; stream buffer
stream_io_str ends

.data
err_disk_full         db  'Disk full',0
err_fopen_mode        db  'Bad mode in fopen',0
err_out_of_handles    db  'Out of stream handles',0
err_stream_handle     db  'Bad stream handle',0

.data?
stream_pointers       dd  FOPEN_MAX dup(?)

.code
extn    free_raw,ms_dos,ms_dos_strerror,set_strerror,malloc_raw
public  fclose,fopen,fread,fwrite

;;      fclose
;
;       entry   BX  stream handle
;       exit    Cf  if error closing file or bad handle
;       uses    AX,BX
;
fclose  proc
        pushm   di,si,es                    ; write bytes left in
        call    fwrite_flush                ;   stream buffer
        jc      fcl1                        ; if disk full

        call    stream_struct_write         ; access stream structure
        jc      fcl1                        ;   if bad handle

        mov     si,bx                       ; clr stream struct pointer
        add     si,si
        add     si,si
        mov     wptr stream_pointers[bp+si],NULL_POINTER
        mov     wptr stream_pointers[bp+si+2],NULL_POINTER

        mov     bx,es:sio_file_handle[di]
        call    free_raw

        mov     ah,3Eh
        call    ms_dos_strerror

fcl1:   popm    es,si,di
        ret
fclose  endp

;;      fopen
;
;       entry   AX  mode, 'r' or 'w' to open for reading or writing
;               DS:SI   filename
;       exit    BX  stream handle
;               Cf  if error
;       uses    AX
;
fopen   proc
        pushm   cx,dx,di,es
        cmp     ax,'r'
        mov     dx,3D00h
        je      fop1                        ; if opening for read only
        cmp     ax,'w'
        jne     fop3                        ; if not open for
        mov     dh,3Ch                      ;   write (unknown mode)
        mov     cx,0
fop1:   mov     ax,dx
        mov     dx,si
        call    ms_dos_strerror             ; msdos with error handling
        jc      fop2                        ; if file open failed
        mov     dx,ax

        call    malloc_stream               ; if no memory or
        jc      fop4                        ;   stream handles

        mov     al,0                        ; zero stream structure
        mov     cx,sio_buffer               ;   except for buffer space
        push    di
        rep     stosb
        pop     di

        mov     es:sio_file_handle[di],dx
        clc
fop2:   popm    es,di,dx,cx
        ret

fop3:   lea     ax,err_fopen_mode           ; *Bad mode in fopen*
        call    set_strerror
        jmp     fop2

fop4:   mov     ah,3Eh                      ; close file, no memory
        mov     bx,dx                       ;   for structure
        call    ms_dos
        stc
        jmp     fop2
fopen   endp

;;      fread
;
;       entry   BX  stream handle
;               CX  byte count
;               ES:DI   destination pointer
;       exit    AX  actual byte count (!=CX for EOF)
;               DI  updated
;               Cf  if error
;
fread   proc
        pushm   di,bx,cx,dx,si,ds
        cmp     bx,stdin
        je      fre3                        ; if special
        call    stream_struct_read          ;   handle, standard input
        jc      fre4
        mov     bx,si
fre1:   jcxz    fre4                        ; if read finished

        mov     ax,wptr sio_position[bx]    ; compute stream
        mov     dx,wptr sio_position[bx+2]  ;   position w/i buffer
        sub     ax,wptr sio_offset[bx]
        sbb     dx,wptr sio_offset[bx+2]
        jnz     fre2                        ; if buffer not
                                            ;   w/i 64k of position
        mov     dx,sio_read_bc[bx]          ; get number of bytes to
        sub     dx,ax                       ;   end of buffer
        jbe     fre2                        ; if stream position
                                            ;   not in buffer
        lea     si,sio_buffer[bx]
        add     si,ax

        call    rep_movsb_limit

        add     wptr sio_position[bx],dx    ; update stream position
        adc     wptr sio_position[bx+2],0
        jmp     fre1

fre2:   call    fread_primitive             ; fill stream buffer
        jbe     fre4                        ;   if error (Cf==1) or EOF
        jmp     fre1                        ;   (Zf==1) else successful

fre3:   mov     ah,3Fh                      ; read from special
        mov     dx,es                       ;   handle stdin
        mov     ds,dx
        mov     dx,di
        call    ms_dos_strerror
        jc      fre4

        add     di,ax                       ; (should not set Cf
                                            ;    unless DI wraps)
fre4:   popm    ds,si,dx,cx,bx,ax
        jc      fre5                        ; if error
        neg     ax                          ;   else return bytes read
        add     ax,di
        clc
fre5:   ret
fread   endp

;;      fread primitive
;
;       entry   DS:BX   stream structure
;       exit    AX  bytes read from file
;               Cf  if error (Zf unknown)
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

you enhance the function, try to give it a better name. I spent at least an hour working on it and **ms_dos_strerror** is the best I could do. If you have any ideas, leave a message on my BBS, and I'll make you famous.

STREAM I/O

The **read_entire_file** procedure from several columns ago provides a very simple mechanism for file input, but it has limitations. The file cannot be too big, the procedure can hog memory, and there is no support for file output.

To correct these problems I adapted the stream I/O functions from the C library. These procedures provide an efficient way to read and write multiple files of arbitrary length. They also are examples of fairly complicated design. You can follow the source code in the magazine or print a real listing.

The stream functions are very similar to the DOS functions for file I/O. In fact, a minimal implementation of the stream functions would take only a few instructions. These stream functions are longer because they support error processing and add local buffering.

The assembly language stream functions and those in the C library handle text mode newlines differently. The C functions delete return characters (0Dh) from return-line feed pairs on input and insert returns before line feeds on output. This is too slow and clumsy for assembly language.

DOS assembly language programs must be intelligent enough to handle end-of-lines with or without return characters. On input, most programs can watch for line feed and ignore return characters. On output, programs can hide the newline sequence behind an equate or procedure call like this:

```
call    putchar_newline  ;write \n to stdout
mov     ax,NEWLINE_CHARS ;write \n to buffer
stosw
```

The STREAM.ASM program begins with a structure declaration. This structure defines the data objects associated with a stream. The structure is a template; all members use question marks in their declaration.

```
eg_str struc     ; sample structure template
   eg_one  db    ?
   eg_two  dw    ?
eg_str ends
```

Assemblers such as MASM accept initialized data objects in structure declarations. You can "expand" these structures in a data segment and access them as locations. I rarely use this facility. In my programs, most structures are in dynamic storage, and I access them with a pointer. The structure members provide the pointer offset and object type.

```
eg_str struc             ; define structure
   eg_one  db    1
   eg_two  dw    2
eg_str ends
.data
    x      eg_str        ; expand structure
.code
    mov    al,x.eg_one   ; access structure
    mov    bx,x.eg_two
```

The structure that is declared in STREAM.ASM is *stream_io_str*, with six members. The first member is *sio_position*. It holds the stream position for the next read or write. When the stream opens, this location is zero. After each **fread** or **fwrite**, this location advances by the number of bytes transferred.

The next member is *sio_offset*. It holds the file position of the first byte in the stream buffer. When the stream functions transfer a chunk of data to the stream buffer, they set this location.

Sio_read_bc holds the input stream buffer byte count. When a function fills the stream buffer, it sets this location to the number of bytes read. *Sio_write_bc* holds the output stream buffer byte count. As the program writes to a stream, this count advances until it reaches the buffer size and the buffer flushes to disk. The current routines do not support read/write access, so an active stream uses only one of the two counting objects.

Sio_buffer is the local stream buffer. Since the buffer size is static, it is OK to define the buffer as a structure member. Had I chosen a dynamic buffer size, I would declare the buffer with an equate instead of as a member.

```
sio_buffer  equ  (size stream_io_str)
```

The next elements in STREAM.ASM are the error strings and stream pointers. The error strings are simple *asciiz* text. Pointers to active streams are in the dword array *stream_pointers*. Each entry contains a 32-bit pointer to a stream structure in dynamic storage or NULL.

The maximum number of open

New Version 3.0

Commenting Disassembler!

SOURCER™ 486

■ **SEE HOW PROGRAMS WORK**
■ **EASILY MODIFY PROGRAMS**

SOURCER™ creates detailed commented source code and listings from memory and executable files directly suitable for reassembly. Built in data analyzer and simulator resolves data across multiple segments and provides detailed comments on interrupts and subfunctions, I/O ports and much more. Includes a definition file facility to include your own remarks and descriptive labels, force data types, and more. Complete support for 8088 through 80486, 8087 to 80387, and V20/V30 instruction sets.

We welcome comparisons with any other product, because no product comes close to the ease of use and output clarity of SOURCER. "Sourcer is the best disassembler we've ever seen." PC Magazine 1/17/89 page 101.

BIOS SOURCE

for PS/2, AT, XT, PC and Clones

■ **CHANGE AND ADD FEATURES**
■ **CLARIFY INTERFACES**

The BIOS Pre-Processor™ with SOURCER provides the first means to obtain accurate legal source listings for any BIOS! Identifies entry points with full explanations. Resolves PS/2's multiple jumps for improved clarity. Provides highly descriptive labels such as "video_mode" and much more. Fully automatic.

ASM ProPak™

■ **ASSEMBLY TOOLS PACKAGE**

Save $50 with the ASM ProPak™ assembly tools package. It includes Sourcer with the BIOS Pre-Processor; Unpacker,™ for unpacking packed EXE files; ASMtool,™ to automatically generate flowcharts and tree diagrams from assembly source code; and View-It,™ to quickly view files of any size and see up to 8 times more information on a single screen using EGA/VGA equipment.

Sourcer–Disassembler	$129.95
BIOS Pre-Processor	49.95
Unpacker–Unpacks packed files	39.95
ASMtool–Source code analyzer	89.95
View-It–Zoomable file viewer	69.95
ASM ProPak–All Above (save $50)	329.75
Sourcer w/BIOS–(save $10)	169.95

Shipping & Handling: USA $5; Canada/Mexico $10; Other Countries $15; CA Residents add sales tax. Visa/MasterCard/COD accepted.

NOT COPY PROTECTED
30-DAY MONEY-BACK GUARANTEE

For orders or information call:

1-800-662-8266

V Communications Incorporated
4320 Stevens Creek Blvd., Suite 275-P
San Jose, CA 95129
(408) 296-4224

Circle **125** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

streams will be set by the equate FOPEN_MAX. This symbol comes from standard C. FOPEN_MAX and symbols for the standard input, output, and error streams are new additions to the ASM.INC include file from a previous column.

The first stream function is **fopen**. If AX is "r," it opens the file in read mode. If AX is "w," it creates the file (write mode). Next, **fopen** allocates a stream structure from dynamic storage and points the first free stream handle to the storage. It then zeros the structure members except for the buffer and stores the DOS file handle in the structure. In case of error, **fopen** sets the error string and closes the file as necessary.

The **fopen** function returns a stream handle in BX. Except for *stdin*, *stdout*, and *stderr*, stream handles are different from DOS handles. A stream handle acts as an index into the *stream_pointers* array.

The next stream function is **fread**. It checks the handle first. If the handle is stdin, **fread** jumps to special code that functions DOS for the read operation. If the stream is not stdin, **fread** accesses the appropriate stream structure with a call to **stream_struct_read** and enters a loop. In **fread**'s main loop, DS:BX points to the stream structure; CX is the master byte count, DX is a count of bytes to end of buffer, SI is a read pointer for the stream buffer, and ES:DI is an output pointer.

The **fread** function enters its main loop at label **fre1**. The JCXZ instruction is both the entry and exit point for the loop. When the master byte count in CX reaches zero, the read operation is done. Using a single exit point for the main loop is slightly slower, but the code is easy to follow.

The code block after **fre1** computes how far the stream position is from the stream buffer offset. Because the flags represent the results of the subtraction of the most significant word, the zero flag indicates if the stream buffer is within 64K of the buffer offset. If it is not, **fread** must refill the buffer.

The next three lines compute how many bytes are available to copy out of the stream buffer. When this number is less than or equal to zero, the JBE instruction branches to code that refills the stream buffer.

The **rep_movsb_limit** procedure acts like a smart REP MOVSB instruction where DX is a secondary byte count. The **rep_movsb_limit** block transfers the minimum number of bytes specified in CX or DX. It then returns an updated count in CX and the number of bytes transferred in DX. After the block transfer, **fread** updates the stream position and starts the loop once again.

When **fread** reaches the end of the stream buffer, it goes to **fread_primitive** for a fill-up. The **fread_primitive** calls DOS to set the position and read the file. The position function should not be necessary; however, it is included

OPTIMAL LINKER TECHNOLOGY

That's right, SLR Systems is THE source for the fastest, most powerful linking technology available. Wherever you are headed, chances are we are there, or working on it. If you don't see what you need here, talk to us.

OPTLINK technology is designed to have no inherent limitations. There is NO internal limit on the number of modules, libraries, publics, segments, etc. The only limit is the amount of disk space available.

OPTLINK/Windows:

Speed: Faster than any other linker, up to 11 times faster than LINK4. Utilizes EMS/XMS before thrashing to disk.

Syntax compatible with LINK4.

Supports Codeview debugging information. Optionally include debug information only from specific modules.

Detailed segment mapping and a complete global cross reference.

Implements FIXDS algorithm to eliminate need for MakeProcInstance, FreeProcInstance, and EXPORTS with EXE files.

Includes OPTIMP, an import library builder. ***Only $250***

Other linkers available include:

Standard **OPTLINK** Generate COM, SYS, and DOS EXE files. Only $150

OPTLINK/Compress Same as OPTLINK but can generate fast self-extract-to-run EXEs 25% to 50% smaller than EXEPACK. Ideal for conserving disk space. Only $250

OPTLINK/2 Complete segmented linker for 16-bit OS/2. Same as OPTLINK/Windows except this linker runs under protected mode OS/2. Only $250

OPTLINK/Locate Generates Intel HEX or OMF output, besides COM, SYS, and DOS EXE. Locate by class and/or segment. Phase for initialized data in rom. Publics and linenumbers in OMF output. Omit classes and/or segments from output. Located CV debugging information to separate file. Only $395

SLR SYSTEMS INC.

1622 N. Main Street, Butler, PA 16001 *(412) 282-0864 Voice* *(412) 282-7965 FAX* *(412) 282-2799 BBS*

Circle 165 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

because I anticipate support for **fseek** in the future.

Positioning the file before each read guarantees proper position and simplifies the code because a test for proper position is unnecessary. The performance penalty from the extra DOS function should be negligible compared to file I/O. After popping the register, **fread** returns the actual number of bytes transferred. This number may not equal CX if the stream is stdin or if end of file occurs.

The next function is **fwrite**. Coding this function cleanly was a struggle. The major complication is that DS is not fully available to access the stream structure. ES is available but using it exclusively would require about twenty ES overrides. I have a programming phobia about many overrides, so I code around the problem.

Function **fwrite** begins with checks for the special handles stdout and stderr. Because these streams are not buffered, **fwrite** calls DOS directly when the program writes to them. Next, **fwrite** calls **fwrite_flush_maybe**. This procedure flushes the stream buffer after a seek. A more efficient approach would flush the buffer in **fseek**, but that must wait until someone writes **fseek**.

Register DS is available to the procedures **fwrite_flush_maybe** and **fwrite_flush**. These procedures use DS to access the stream structure and suffer a small performance penalty from push/pop instructions and the call to **stream_struct_read**. Notice that I intentionally trade speed for clarity in this case.

In **fwrite**'s main loop, the registers are assigned as follows: AX is the stream buffer byte count; BX is the stream handle; CX is the master byte count; and DX is the count of free space in the stream buffer. DS:SI serves as the input pointer while ES:DI points to the stream structure.

The **fwrite** function uses a JCXZ instruction as the entry/exit point for its main loop (ignore the CLC). The block of instructions following JCXZ computes the space available in the stream buffer. The JB branch is insurance against a program gone haywire because the **fwrite** function never leaves a full buffer.

The next code block copies data to the stream buffer. The code uses DI as a destination pointer within the block. The byte count in DX is used to update the stream byte count and the stream position.

Before starting the loop again, **fwrite** checks the stream buffer byte count. If the buffer is full, **fwrite** calls **fwrite_flush** to write the buffer to disk, update the buffer offset, and clear the buffer byte count. When the master byte count in CX reaches zero, the loop is exited. An **fwrite** call restores the registers and returns the number of bytes written.

To release an open stream, call **fclose**. This function writes any remaining bytes of a stream buffer to an output file, frees the stream handle, releases storage for the stream structure, and closes the file.

The single character stream functions are easy to create from the block routines. A **fputc** follows. It pushes AX to create a one-character buffer on the stack, points DS:SI to that byte, and calls **fwrite**.

```
;;      fputc
;
;       entry   AL      character
;               BX      stream handle
;       exit    Cf      if error
;
fputc   proc
        pushm   cx,si,ds,ax
        mov     ax,ss
        mov     ds,ax
        mov     si,sp
        mov     cx,1
        call    fwrite
        popm    ax,ds,si,cx
        ret
fputc   endp
```

After you finish writing an important module, make a printed listing and review each line. There is no better way to improve code quality. During the review, you should:

- Verify entry/exit conditions in header comments.
- Fill in any missing instruction comments.
- Look for obvious optimizations.
- Compare PUSH and POP lists.
- Find bugs by mentally tracing all instructions.
- Look for bugs related to boundary conditions.
- Ensure that procedure calls assume the correct register entry and exit conditions.

Use a pencil to mark corrections directly on the listing. Put a small circle for

AMX™

REAL-TIME MULTITASKING KERNEL

8086/88, 80x86/88 80386
Z80, 64180, 8080/85 68000/10/20

- Fast, reliable operation
- Compact and ROMable
- PC peripheral support
- DOS file access
- C language support
- Preemptive scheduler
- Time slicing available
- Configuration Builder
- Complete documentation
- Intertask messages
- Message exchanges
- Dynamic operations
  - — task create/delete
  - — task priorities
  - — memory allocation
- Event Manager
- Semaphore Manager
- List Manager
- InSight™ Debugging Tool

THE BEST

Join over 600 developers such as IBM®, Xerox, Hewlett Packard, Hayes, Hughes Aircraft and NASA.

CHOOSE AMX

The best low-cost, high-performance real-time multitasking system available today.

No Royalties
Source Code Included

Demo Disk	**$25 US**	*Call for prices for other processors.*
Manual only	**$75 US**	
AMX 86	**$3000 US**	

(Shipping/handling extra)

IBM is a registered trademark of IBM Corp.
Z80 is a trademark of Zilog, Inc.
AMX, AMX 86, InSight are trademarks of KADAK Products Ltd.

KADAK Products Ltd.
206-1847 West Broadway
Vancouver, B.C., Canada
V6J 1Y5
Telephone: (604) 734-2796
Fax: (604) 734-8114

Circle 211 on Reader Service Card

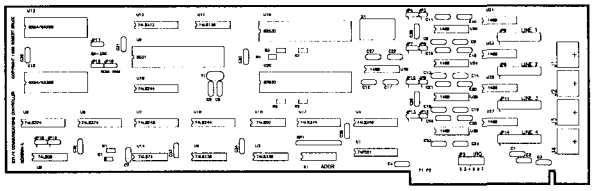
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;               Zf  if end of file
;       uses    SI
;
fread_primitive proc
        pushm   bx,cx,dx
        mov     si,bx

        mov     ax,4200h                        ; position file
        mov     bx,sio_file_handle[si]
        mov     dx,wptr sio_position[si]
        mov     cx,wptr sio_position[si+2]
        call    ms_dos_strerror
        jc      frp1                            ; if error position file

        mov     wptr sio_offset[si],ax
        mov     wptr sio_offset[si+2],dx
        mov     sio_read_bc[si],0

        mov     ah,3Fh                          ; read file
        mov     cx,size sio_buffer
        lea     dx,sio_buffer[si]
        call    ms_dos_strerror
        jc      frp1                            ; if read error
        mov     sio_read_bc[si],ax

        or      ax,ax                           ; set Zf for end of file
frp1:   popm    dx,cx,bx
        ret
fread_primitive endp

;;      malloc stream
;
;       exit    BX  stream handle
;               ES:DI   stream structure
```

```
;               Cf  if no handles or memory left
;       uses    AX
;
malloc_stream proc
        pushm   cx,si
        mov     bx,stderr                       ; find free stream handle
mas1:   inc     bx
        cmp     bx,FOPEN_MAX
        jae     mas3                            ; if no handles left

        mov     si,bx
        add     si,si
        add     si,si
        les     di,stream_pointers[bp+si]
        mov     ax,es
        or      ax,di
        jnz     mas1                            ; if stream allocated

        mov     cx,size stream_io_str           ; allocate memory for
        call    malloc_raw                      ;   stream structure
        jc      mas2                            ; if no memory

        mov     wptr stream_pointers[bp+si],di
        mov     wptr stream_pointers[bp+si+2],es

mas2:   popm    si,cx
        ret

mas3:   lea     ax,err_out_of_handles           ; *Out of stream handles*
        call    set_strerror
        jmp     mas2
malloc_stream   endp

;;      rep movsb limit
```


Intelligent Four Channel

Serial Communications Controller



- 8031 microcontroller
- 82530 SCCs
- 32K memory
- RS-232
- RJ-45 connectors
- DOS device driver
- FCC Class A approved
- Excellent documentation
- BASIC, C and Pascal software source examples
- **$495 complete** with all cables, software, & extensive documentation

Designed, Manufactured, Sold and Serviced by:

RTech Inc.

Voice . . . (612) 345-4555
FAX . . . . (612) 345-5514
Modem . . (612) 345-4656

907 N. 6th St., Lake City, MN 55041

Total control with *LMI FORTH*™

For Programming Professionals: an expanding family of compatible, high-performance compilers for microcomputers

For Development:
Interactive Forth-83 Interpreter/Compilers for MS-DOS, OS/2, and the 80386

- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 500 page manual written in plain English
- Support for graphics,floating point, native code generation

For Applications: Forth-83 Metacompiler

- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, 6303, 6809, 68HC11,34010, V25, RTX-2000
- No license fee or royalty for compiled applications

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone Credit Card Orders to: (213) 306-7412
FAX: (213) 301-0761

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;
;       entry   DS:SI   source pointer
;               ES:DI   destination pointer
;               CX  byte count
;               DX  byte count limit
;       exit    SI,DI   updated
;               CX  updated (nonzero if original DX<CX)
;               DX  actual number of bytes transfered (minimum of CX,DX)
;
rep_movsb_limit proc
        push    ax
        cmp     cx,dx                   ; DX is unsigned minimum of CX and DX
        xja     rmm1
        mov     dx,cx

rmm1:   mov     ax,cx                   ; this procedure implements a smart
        sub     ax,dx                   ;  "rep movsb" instruction.  the
        mov     cx,dx                   ;  difference is it moves the minimum
        rep     movsb                   ;  number of bytes specified in CX or
        mov     cx,ax                   ;  DX. CX is updated normally; DX has
        pop     ax                      ;  the actual number of bytes moved.
        ret
rep_movsb_limit endp

;;      stream struct read
;
;       entry   BX  stream handle
;       exit    DS:SI   stream structure
;               Cf  if bad handle
;       uses    AX
;
stream_struct_read proc
        cmp     bx,FOPEN_MAX
        jae     rss1                    ; if handle too large
        mov     si,bx
        add     si,si
        add     si,si
        lds     si,stream_pointers[bp+si]
        mov     ax,ds
        or      ax,si
        jz      rss1                    ; if inactive stream
        ret

rss1:   lea     ax,err_stream_handle    ; *Bad stream handle*
        jmp     set_strerror
stream_struct_read endp

end
```

LISTING 2

```
include asm.inc

.stack
.data
extrn           dgroup_segment:word

bytes_text      db  ' bytes',0
premature_text  db  'Premature EOF',0
reading_text    db  'Reading ',0
removing_text   db  'Removing ',0
writing_text    db  'Writing ',0

fname_text      db  'TEST_SIO.TMP',0

.data?
stream_data     db  10000 dup(?)
```

continued

FREE ISSUE!

PJ readers are invited to receive a FREE sample issue of Midnight Engineering.

The Journal Of Personal Product Development provides unique mix of practical hardware and software engineering mixed with the business aspects of small scale product development. Your guide to developing and marketing your own products.

Midnight Engineering
Suite 7041
111 E. Drake Rd.
Ft. Collins, CO 80525
303-225-1410 (anytime)

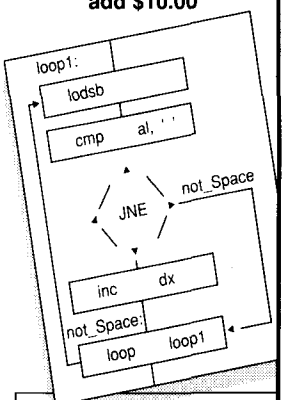
Circle **272** on Reader Service Card

ASMFLOW $99.95

S/H $3.00 Outside U.S. add $10.00

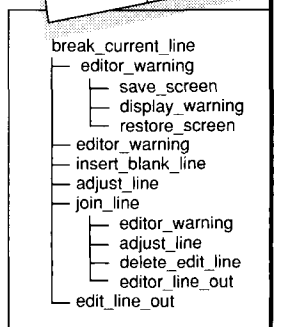
AT LAST!

YOU CAN STEP BACK AND TAKE A LOOK AT YOUR CODE



FLOW CHART AND ANALYZE YOUR ASSEMBLY LANGUAGE SOURCE CODE

- Flow Charts
- Tree Diagrams
- Stack Sizing
- Register Analysis
- CPU Timing Analysis
- Procedural X-Reference
- 8088/87 to 80386/387
- Context-Sensitive Help
- Menu/Batch/Command line Operation
- MASM 5.1 Compatible



QUANTASM CORPORATION
19855 Stevens Creek Blvd, Suite 154
Cupertino, CA 95014
(408) 244-6826

VISA • MC 30 - Day Money Back Guarantee

Circle **252** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

each change along the right edge of the listing. When you modify the source file, check off the circle.

The final tasks after desk checking are testing and debugging. A thorough testing should find the obvious programming errors left after desk checking and most of the difficult bugs. Many of the hard-to-find bugs occur in the boundary conditions.

BOUNDARY TESTS

Boundary conditions occur when a data object is near some limit or special number. A parameter might equal zero or minus one; a buffer might fill exactly. Applications are rarely good test programs since they never exercise boundary conditions until you ship the program to a customer and chaos ensues. The quick way to find such bugs is to write a test program specifically designed for boundary conditions.

TEST_SIO.ASM is a test program for the stream functions. It starts by deleting a nonexistent file. If an error returns, TEST_SIO calls **perror** to display the current error string. TEST_SIO then creates a file and writes four chunks of data of differing lengths. The first chunk is exactly one buffer length. The second is one byte less than a buffer length. The third chunk is two bytes longer, while the fourth is exactly one buffer length again.

TEST_SIO next closes the file and reopens it for input. It reads back the same four buffer lengths and closes the file once again. Before returning to DOS, TEST_SIO deletes the file.

TEST_SIO is not thorough enough. It should read and write blocks of length 0, 8000h, and 0FFFFh. It should open files for reading and writing in such a way that the functions fail with errors including no memory, no handles, and no such file. It should write to a file until the disk is full. That would be a good test program.

Do not just execute the test program. Use a debugger to single-step the program and watch it carefully. The test may trigger a bug without any external effect.

WRAPPING UP

Are you tired of reaching to the back of the magazine rack for your copy of *Programmer's Journal*? I am. Do me a favor. Move a few copies of *PJ* up to the front of the rack. While you're at it, go ahead and cover up a copy of *Byte* or *PC Magazine!*

I have not seen the new version of TASM or MASM. TASM is due in a few weeks, but don't hold your breath waiting for MASM. According to unofficial sources at Microsoft, the target date is "around" the end of this year. I did get Microsoft C 6.0. I think the manuals are a big improvement over 5.1. I have yet to try the software.

Be sure to call my BBS (213-559-1449) and leave me your comments. The files for this column are **pj84.zip** and **pj84.arc**. PJ

```
.code
extn    startup,puts,remove,perror,exit_program,put_string
extn    fopen,fclose,fread,fwrite,put_unsigned

;;      main
;
main    proc
        mov     ax,@data            ; compensate for MS-LINK bug
        mov     ss,ax
        mov     sp,offset stack

        mov     ax,sp               ; initialize segments, argv, argc, etc
        call    startup
        mov     es,dgroup_segment[bp]
        mov     ds,dgroup_segment[bp]

        lea     si,removing_text    ; *Removing TEST_SIO.TMP*
        call    put_string
        lea     si,fname_text
        call    puts

        call    remove              ; delete TEST_SIO.TMP
        jnc     mai1                ;   if deleted (should not happen)
        call    perror              ;   else display error (test perror)
... test_write call omitted for space
        lea     si,reading_text     ; *Reading TEST_SIO.TMP*
        call    put_string
        lea     si,fname_text
        call    puts

        mov     ax,'r'              ; open TEST_SIO for input
        call    fopen
        jc      mai3

        mov     cx,8192             ; read various data lengths
        call    test_fread          ;   equal, less, and greater than
        mov     cx,8192-1           ;   internal stream buffer size
        call    test_fread
        mov     cx,8192+2
        call    test_fread
        mov     cx,8192
        call    test_fread
        call    fclose
        jc      mai2
        lea     si,fname_text       ; delete TEST_SIO.TMP
        call    remove
        jmp     exit_program
mai2:   mov     si,NULL_POINTER
mai3:   call    perror
        jmp     exit_program
main    endp

;;      test fread
;
;       entry   BX  stream handle
;               CX  byte count
;               DS,ES   DGROUP
;       uses    AX,DI,SI
;
test_fread proc
        mov     ax,cx
        call    put_unsigned
        lea     si,bytes_text
        call    puts
        lea     di,stream_data
        call    fread
        jc      tfr1
        cmp     ax,cx
        jne     tfr2                ; if premature EOF
        ret
tfr1:   mov     si,NULL_POINTER
        call    perror
        jmp     exit_program
tfr2:   lea     si,premature_text
        call    puts
        jmp     exit_program
test_fread endp
... test_fwrite proc omitted for space
end     main
```

All code in this issue is available through our bulletin board services or directly from *PJ* on disk. For more information see pages 8 and 88.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

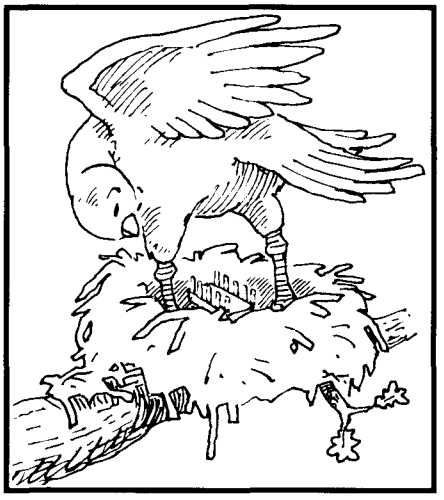
Subprograms in COBOL using the CALL verb

COBOL-85 nested

By Richard D. Riehle

One of COBOL-85's more interesting innovations is the enhancement of the CALL verb and its associated options. The COBOL-85 standard implements the two software engineering principles of modularity and localization (see *PJ* Vol 6.5, "Ada, A Software Engineering Tool").

At last COBOL programmers can do many of the things they are accustomed to doing in Pascal, Ada, C, and QuickBasic, including coming close to creating loosely-coupled modules in COBOL.



The program fragment in Listing 1 demonstrates how a programmer might nest one program within another. In this case, TRAN-VALIDATION is nested within MASTER-FILE-UPDATE. It is called with two parameters, one by reference and the other by content. The parameters are positional and "type" compatible; i.e., the parameter **ws-tran-record** matches **tran-record**, while **tran-rec-status** matches **result-code**.

The "by reference" parameter means that any change to **result-code** will automatically change **tran-rec-status.** "By content" means that any change to the matching parameter, **tran-record**, must be local to the subprogram, TRAN-VALIDATION, and not affect **ws-tran-record**; "by content" protects **ws-tran-record** from being modified by TRAN-VALIDATION.

This is the way COBOL programs should be designed under COBOL-85. It requires more careful program planning, more managerial oversight, and greater programmer understanding of the concepts of software engineering. The loosely-coupled module approach becomes especially important as the use of CASE (Computer Assisted Software Engineering) tools increases.

The use of parameterized subprograms is not new to users of other languages. COBOL-85 now makes it an appropriate alternative to the traditional parameterless PERFORM statement. The PERFORM has been dependent on accurately defined (and spelled) GLOBAL data. In CALLed subprograms, data definition is local to the subprogram, so a data name in one subprogram will not conflict with an identical data name in some other subprogram. The general rule in future COBOL systems is: avoid the use of PERFORM for invocation of any subroutine; CALL a nested subprogram instead.

Properly designed COBOL systems will make use of the COPY and COPY... REPLACING statements to maintain the subprogram modules in reusable code libraries. Use of libraries is also possible because of the localization of data name definition.

INITIAL

The use of the word INITIAL in the PROGRAM-ID of the nested program ensures module reinitialization. A COBOL subprogram is, by default, "state persistent," meaning that it will remain in the state of its last execution unless re-initialized. The program can either "CANCEL" (reinitialize) the module prior to each CALL, or code the module as INITIAL.

PC IMPLEMENTATIONS

There are some excellent implementations that support the ANSI COBOL-85 standard for microcomputer systems under MS-DOS and OS/2. Two of the best, in my opinion, are Micro-

Richard D. Riehle is an instructor in Ada at Merritt College in Oakland and at Foothill College in Los Altos, California. He is also an independent computer consultant with over twenty years in software development. Mr. Riehle's training organization, ICE (Intensive Computer Education), conducts seminars on a variety of computer subjects. Readers may contact Richard at 1090-105 Tanland Drive, Palo Alto, CA 94303, (415) 858-1267.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Focus (which is now the foundation for Microsoft COBOL) and Ryan-McFarland RM-COBOL. MicroFocus appears to have some interesting non-standard extensions that I have not used because of portability concerns. MicroFocus has had the good manners to document which features are not portable and which features are portable to other environments. As of this writing, my preferred implementation of an MS-DOS COBOL is the one from MicroFocus.

AN ILLUSTRATIVE EXAMPLE

Our example program contains some unusual code. The EVALUATE statement demonstrates how to set up a decision table using COBOL-85. This is similar to the traditional CASE statement in other languages, but far more powerful. In fact, the EVALUATE statement in COBOL is one of the most powerful CASE statements to be found in any programming language. Notice that the nested programs must each terminate with an EXIT PROGRAM statement and have an END PROGRAM *program-id* as its last statement and scope delimiter. The main program must have an END PROGRAM *program-id* statement terminating its scope, and this statement must be the final statement in the code.

CALL by content does not permit a literal value, an expression, or the name of another subprogram as a parameter. The parameter must be a data name.

As an exercise to test how CALL by content can work, try the program in Listing 2 with your COBOL-85 compiler. Be sure to make a careful note of which values change, and where they change.

This little program should succinctly demonstrate the difference between "by content" and "by reference." It might even tickle your imagination and enhance the quality (and modularity) of your COBOL code on future projects. PJ

LISTING 1

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MASTER-FILE-UPDATE.

   ...

WORKING-STORAGE SECTION.
01 WS-TRAN-RECORD.
    05 TRAN-CODE           PIC X(02).
    05 TRAN-VALUE          PIC S9(3)V99.
    05 TRAN-DATE           PIC X(06).
01 PROGRAM-STATUS-CODES.
    05 TRAN-REC-STATUS     PIC S9(02) COMPUTATIONAL.
    05 MAST-REC-STATUS     PIC S9(02) COMPUTATIONAL.

PROCEDURE DIVISION.
    CALL TRAN-VALIDATION
                    USING BY CONTENT WS-TRAN-RECORD,
                          BY REFERENCE TRAN-REC-STATUS.
    IF TRAN-REC_STATUS NOT = ZERO THEN
                    [ do some statements ]
    ELSE
                    [ do some other statements ]
    END-IF

    ... [ insert more code here ] ...

    STOP RUN.
IDENTIFICATION DIVISION.
PROGRAM-ID. TRAN-VALIDATION INITIAL.
...

WORKING-STORAGE SECTION.
01 VALIDATION-VALUES.
    05 DATE-STATUS         PIC 9(01) VALUE ZERO.
    05 VALUE-STATUS        PIC 9(01) VALUE ZERO.
    05 TRAN-CODE-STATUS    PIC 9(01) VALUE ZERO.
   ...
LINKAGE SECTION.
01 TRAN-RECORD.
    05 TRAN-CODE           PIC X(02).
    05 TRAN-VALUE          PIC S9(03)V99).
    05 TRAN-DATE           PIC X(06).

01 RESULT-CODE             PIC S9(02) COMPUTATIONAL.

...

PROCEDURE DIVISION USING
                        TRAN-RECORD,
                        RESULT-CODE.
000-MAIN.
    PERFORM DATE-VALIDATION.
    PERFORM TRAN-CODE-VALIDATION.
    PERFORM VALUE-VALIDATION.

    EVALUATE DATE-STATUS      = ZERO
             ALSO TRAN-CODE-STATUS = ZERO
             ALSO VALUE-STATUS     = ZERO
            WHEN TRUE  ALSO TRUE  ALSO TRUE  MOVE ZERO TO RESULT-CODE
            WHEN FALSE ALSO TRUE  ALSO TRUE  MOVE -1   TO RESULT-CODE
            WHEN FALSE ALSO FALSE ALSO TRUE  MOVE -2   TO RESULT-CODE
            WHEN FALSE ALSO FALSE ALSO FALSE MOVE -3   TO RESULT-CODE
            WHEN TRUE  ALSO TRUE  ALSO FALSE MOVE -4   TO RESULT-CODE
             WHEN OTHER                      MOVE -10  TO RESULT-CODE
    END-EVALUATE

    ... [ other program statements ] ...

    EXIT PROGRAM.
END PROGRAM TRAN-VALIDATION.
END PROGRAM MASTER-FILE-UPDATE.
```

LISTING 2

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DISPLAY-IT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 DISPLAY-VALUES.
    05 VALUE-1    PIC 9(03) VALUE 5.
    05 VALUE-2    PIC 9(03) VALUE 5.
PROCEDURE DIVISON.
001-MAIN-ROUTINE.
    DISPLAY "INITIAL VALUE-1 = " VALUE-1.
    DISPLAY "INITIAL VALUE-2 = " VALUE-2.
    CALL VALUE-CALCULATOR
                USING  BY REFERENCE VALUE-1,
                       BY CONTENT VALUE-2.
    DISPLAY "    NEW VALUE-1 = " VALUE-1.
    DISPLAY "    NEW VALUE-2 = " VALUE-2.
    STOP RUN.
IDENTIFICATION DIVISION.
PROGRAM-ID. VALUE-CALCULATOR INITIAL.
DATA DIVISION.
LINKAGE SECTION.
01 CALCULATION-VALUES.
    05 CALC-VALUE-1  PIC 9(03).
    05 CALC-VALUE-2  PIC 9(03).
PROCEDURE DIVISION
                 USING  CALC-VALUE-1,
                        CALC-VALUE-2.
001-CALCULATION-ROUTINE.
    ADD 1 TO CALC-VALUE-1.
    ADD 1 TO CALC-VALUE-2.
    DISPLAY "CALC VALUE-1 = " CALC-VALUE-1.
    DISPLAY "CALC VALUE-2 = " CALC-VALUE-2.
999-ALL-DONE.
    EXIT PROGRAM.
END PROGRAM VALUE-CALCULATOR.
END PROGRAM DISPLAY-IT.
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

"Quick and Clean®" COBOL

COBRA® – THE COBOL REPORT ACTIVATOR SYSTEM

What Is the COBRA System?

The COBOL REPORT ACTIVATOR (*COBRA*) System is a simple, quick and powerful method used to generate error free structured COBOL programs. It is an ideal tool for ad-hoc reports, file reformatting and multiple program systems. *COBRA* may be used with most COBOL compilers.

Because the *COBRA* generated COBOL code is structured and well written, it is easy to follow and change. Multiple program systems are easily maintained. Reports created by the *COBRA* system may vary based on user demand, but will contain standardized features. Control break and total reports are easily produced, files may be automatically sorted, and records may be selected only when specific conditions exist.

Each time a *COBRA* system report is created, an error-audit report is also created. It shows statistics on records read, selected, rejected because of invalid data, number of detail lines printed, and when necessary, the number of output records for each file. The first 80 characters of invalid records are printed, along with a message giving the reason for rejection.

How does the COBRA System work?

The *COBRA* system is a 'keyword driven' system - that is, each statement begins with a keyword. It is also freeform; any statement may start in any column, and statements may be in any order. The COBOL code is generated based on your selection of keywords.

List Price $495 • For more information please contact:

COBRA Systems Software

505 Acadia Drive P.O. Box 750817
Petaluma, CA 94975 • (707) 763-5123

The COBRA System:

* Is easy to learn and use
* Produces efficient structured COBOL code
* Uses saved file descriptions
* Accepts your COBOL code and COPY members
* Can create five output files per program
* Provides for record selection
* Sorts in ascending, descending, or mixed sequences, or, has a no-sort option
* Supports Indexed and Sequential files
* Provides extra report options, including:
  - printing the TIME the report was produced
  - putting dashes above and/or below column headings and before the final report line
* Allows site defined defaults for efficiency and standards, including:
  -Input record length up to 9995 characters
  -Report page width up to 160 characters
  -Report spacing flexibility (single, double, or triple spaced)
  -Report date format
  -Whether to print *COBRA* source statements in the generated COBOL code
* Offers a money-back guarantee

Circle **281** on Reader Service Card

HARVARD SOFTWORKS

NUMBER ONE IN FORTH INNOVATION

(513) 748-0390 P.O. Box 69, Springboro, OH 45066

MEET THAT DEADLINE ! ! !

- Combine the raw power of extensible languages with the convenience of carefully implemented functions!
- Execute faster than optimized C!
- Compile 40,000 lines per minute!
- Stay totally interactive, even while compiling!
- Program at any level of abstraction from machine code thru application specific language with equal ease and efficiency!
- Modify routines without recompiling!
- Use source code for 2500 functions!
- Use data structures, control structures, and interface protocols from any other language!
- Implement that borrowed feature, often more efficiently than in its source!
- Use an architecture that supports very small programs or full megabyte ones with a single version!
- Forget chaotic syntax requirements!
- Use subroutine libraries written for other languages! More efficiently!
- Outproduce experienced programmers stuck using conventional languages!

HS/FORTH with FOOPS - The only full multiple inheritance object oriented language under MSDOS!

WAKE UP ! ! !

Forth is no longer a language that tempts programmers with "great expectations", then frustrates them with the need to reinvent simple tools expected in any commercial language.

HS/FORTH Meets Your Needs!

We don't shortchange you with promises. We provide implemented functions to help you complete your application quickly.

Don't judge Forth by public domain products or ones from vendors primarily interested in consulting - they profit from not providing needed tools! Public domain versions are cheap - if your time is worthless. Useful only in learning Forth's basics, they fail at showing its true potential. Not to mention being s-l-o-w.

You can't add extensibility to fossilized compilers. You can easily add features from any other language to **HS/FORTH**. Try our full modifiable multiple inheritance **OOPs** extension or our **Fortran** translator & mathpak!

HS/FORTH has more features and libraries than we can possibly describe here. **Please ask for our brochure!**

HS/FORTH uses MSDOS or PCDOS. Each level includes features of lower ones. Level upgrades: $25. plus price difference. Sources in ordinary text files.

STUDENT LEVEL $145.
text & scaled/clipped graphic windows, ellipses, splines, arcs, fills, turtles; powerful parsing, formatting, file and device I/O; shells; interrupt handlers

PERSONAL LEVEL $245.
software floating point, trig, transcendental, quad & scaled integer, automatic optimizer for assembly like performance.

PROFESSIONAL LEVEL $395.
hardware floating point, turnkey, seal, Rosetta Stone Dynamic Linker, round robin & time slice multitaskers, dynamic string manager, file blocks, sector mapped blocks, assemblers

PRODUCTION LEVEL $495.
Metacompiler: DOS/ROM/direct/indirect C data structures & struct compiler, TurboWindow-C graphics library,

Options:

FOOPS+ with multiple inheritance	**$ 75.**
286FORTH or **386FORTH**	**$295.**
TurboWindow-C (Metagraphics)	**$ 99.**
BTRIEVE for HS/FORTH (Novell)	**$199.**
ROMULUS HS/FORTH from ROM	**$ 95.**
FFORTRAN translator/mathpak	**$ 75.**

Circle **42** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ PRESSROOM

ACTOR BREAKS 640K BARRIER

The Whitewater Group has released Actor 3.0, a major enhancement to one of the most popular object-oriented development systems for the Windows environment. Actor 3.0 was designed specifically for Microsoft Windows version 3.0 and takes full advantage of its features. Applications developed in Actor will run in any configuration of Windows 3.0, including real and protected modes.

Actor 3.0 includes an automatic memory swapping system that enables the user to create programs of up to two megabytes in size, yet which will still run on a 640K system. Actor allows the developer to create professional Windows applications with significantly less code because it provides built-in classes of objects for windowing, creating dialog boxes and graphics, data management, and printing. Actor 3.0 sells for $695 (upgrades are $75).

The Whitewater Group Inc.
1800 Ridge Avenue
Evanston, IL 60201
(800) 869-1144

VOICE SUPPORT IN ARTISOFT

Artisoft's version 3.0 of the LANtastic Network Operating System (NOS) supports voice chat and voice mail when LANtastic Voice Adapters are installed on each PC. With NOS 3.0 and the new voice adapter (purchased separately), users can pick up the telephone handset provided with each adapter and leave voice messages or talk with others on the network. Version 3.0 also offers a textual chat option, a pop-up message that informs the user when someone wants to chat textually or verbally or when a text or voice message is waiting.

NOS 3.0 also offers a disk cache, easier menu-driven installation, support for diskless workstations, printing speed improvements, additional security, more open files per server, and context-sensitive help.

Artisoft has also announced the LANtastic Voice Programmer's Interface (VPI). The VPI is a set of simple software commands that interface with the LANtastic Voice Adapter, allowing developers to add high-fidelity sound effects, including speech and music, to software programs. Developers can, for example, now create educational and training programs on disk or CD-ROM to give verbal instructions to students, send voice mail, and make talking software demos using the voice adapter.

Each $149 IBM-compatible 8-bit LANtastic Voice Adapter comes with a telephone handset and cord that connects it to the adapter. The LANtastic VPI is priced at $195.

Artisoft Inc.
575 E. River Road
Tucson, AZ 85704
(602) 293-6363

B+ TREES FOR QUICKBASIC

CDP Consultants announces Index Manager (IM), an easy-to-use file-indexing system for QuickBASIC. Programmers can create B+ tree file indexes within QuickBASIC programs and files can be accessed randomly by full key, browsed by partial key, or sorted forward or backward. All of the IM functions are performed by one external subroutine. Only index files are managed; the programmer retains full control over all data files. The Index Manager package costs $59. A demo version is also available.

CDP Consultants
1700 Circo Del Cielo Drive
El Cajon, CA 92020
(619) 440-6482

PC-LINT 4.0 FOR DOS AND OS/2

Gimpel Software has released Version 4.0 of PC-Lint, a C source code analysis tool that runs under MS-DOS or OS/2. This new version adds more than 70 new diagnostic messages, including checks for compile-time objects such as typedefs, macros, and declarations.

PC-Lint reports whether the items are used locally or globally. It will also point out header files that were not used, externals that can be made static, and declarations that can be removed from header files.

PC-Lint 4.0 supports the notion of a lint object module which captures all the external information of a module or modules in binary form. This allows incremental linting via a MAKE file. PC-Lint is available for $139.00 and comes with a 30-day, money-back guarantee.

Gimpel Software
3207 Hogarth Lane
Collegeville, PA 19426
(215) 584-4261

GSS ENHANCES TOOLS

Graphic Software Systems (GSS) has brought out enhanced versions of its Graphics Development Toolkit (GDT) and Graphical Kernel System (GKS) for DOS. GDT Release 2.16 and GKS Release 2.13 feature expanded device and font support. Enhanced drivers in these products can increase printing speed by as much as a factor of 30 for certain applications.

Devices added to the GDT and GKS toolkits include the Toshiba T5100 portable, Headland (Video Seven) VGA cards, Epson EPL-6000 and Tektronix ColorQuick Ink-jet printers. In addition, Computer Graphics Metafile (CGM) and AutoCAD's Drawing Interchange File (DXF) formats are supported. The new releases also include updated Bitstream Fontware software to support soft fonts.

The GDT now provides a C language binding for Rational System's 16/M protected-mode DOS extender. Large applications have the advantage of being able to use DOS/16M's protected mode memory. GDT and GKS are priced at $595 each, and upgrades are available.

Graphic Software Systems, Inc.
9590 SW Gemini Drive
Beaverton, OR 97005
(503) 641-2200

WARPLINK OVERLAY LINKER

Hyperkinetix has released Warplink, a Microsoft-compatible dynamic overlay linker. It is intended for use with any compiler that produces Microsoft-compatible object files. Warplink is specifically designed to support Microsoft C and MASM, Turbo C and TASM, and Force, Clarion, and Clipper database compilers. It also works with Hyperkinetix's own BUILDER batch file compiler.

Warplink's error handling offers diagnosis of linking errors and suggests solutions for these errors. The software provides detailed map information to help with debugging and comes with an

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

option to create a COM file directly. Warplink is priced at $249.95.

Hyperkinetix, Inc.
666 W. Baker Suite 405
Costa Mesa, CA 92626
(800) 873-9993

C WINDOWS FOR 386

Magna Carta Software has released a new version of its C Windows Toolkit that supports WATCOM C/386. C Windows Toolkit is a full-featured user interface library that permits C programmers to quickly add text windowing, help screens, and pop-up and spreadsheet-style menus to applications.

C Windows Toolkit already supports DOS C compilers from Watcom, Borland, and Microsoft. The newly-announced support for WATCOM C/386 means that C programs can now execute in 386 protected mode incorporating the C Windows Toolkit.

WATCOM C/386 requires a 386 DOS extender and tools from either Phar Lap Software or Ergo Computing. (The present version of C Windows Toolkit for WATCOM C/386 works with the Phar Lap tools; support for Ergo's OS/386 tools will follow.) The WATCOM C/386 version of C Windows Toolkit is $199.90 (upgrades are $99.95).

Magna Carta Software
P.O. Box 475594
Garland, TX 75047-5594
(214) 226-6909

WINDOWS 3.0 ARRIVES

Microsoft Corp. has released Microsoft Windows version 3.0. This new version incorporates better microprocessor utilization, improved memory management, network compatibilities, a user interface similar to OS/2 Presentation Manager, and a number of new or updated applications.

Version 3.0 supports three modes: real, standard, and 386 enhanced. The *real* mode is for computers with less than 1 megabyte of memory or machines with 8086 microprocessors. The *standard* mode requires at least a megabyte of memory and a 286 or a 386 CPU. Windows will then run in the protected mode and can directly access up to 16 MB of memory. The *386 enhanced* mode requires 2 MB of extended memory and uses the virtual-8086 mode to provide preemptive multitasking to the applications. A demand-paged virtual memory manager provides applications with up to 48 MB of virtual memory.

Windows decides which mode to use during start-up by examining the hardware available. The mode may be overridden from the command line. The 386 enhanced mode also provides switching between non-Windows applications using a single keystroke combination. Priority for background processes can also be set using the new Control Panel.

Memory management improvements provide major benefits to users in a local area network environment. In addition, Microsoft has built in "network awareness" throughout Windows. For example, the Windows Setup program makes it easy to install and configure Windows on network stations.

Better display hardware support, a proportionally-spaced system font, 3-D scroll bars, command buttons and colorful icons make Windows more attractive and appealing. One of the new applications included is the File Manager. The File Manager provides standard file manipulation as well as a mouse interface that allows files and directories to be moved and copied by dragging the mouse.

All Windows 2.x applications will run under Windows version 3.0 in the *real* mode. Windows applications must be identified by the vendor as Windows 3.0 compatible to be used in the other modes. Microsoft Windows 3.0 is available for $149 (upgrades are $50).

Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052-6399
(206) 882-8080

DB_VISTA SUPPORTS DLLS

Raima Corporation has announced that its db_VISTA III Database Management System now supports Microsoft Windows' Dynamic Link Libraries (DLLs) with its database engine. The ability to dynamically link library code into an application is an important feature of Windows, allowing multiple applications to share code segments. This results in a major savings of memory. db_VISTA now includes a DLL and supports Windows' memory management functions.

Raima Corporation
3245 146th Place
Bellevue, WA 98007
(800) 327-2462

PANEL PLUS II'S 2.1

Roundhill Computer Systems is announcing its new 2.1 version of the PANEL Plus II screen manager and screen library. This release incorporates changes that significantly enhance conformance to the Common User Access (CUA) specification of IBM's Systems Application Architecture (SAA).

Version 2.1 of PANEL Plus II adds full support for Microsoft C 6.0. It includes libraries for all memory models of MS-C 6.0 and dynamic link libraries for use with MS-C 6.0 under OS/2. An on-line help database containing details of all PANEL Plus II library functions is also provided. This database works with Microsoft's QuickHelp utility and can be accessed from the Programmer's Workbench in MS-C 6.0.

Other enhancements in this new release include full word-wrap text editing, automatic conversion of currency and date formats to different national standards, and support for 80386 C compilers from Watcom and MetaWare with Phar Lap, Ergo, and Rational Systems DOS extenders.

PANEL Plus II is available on many C development platforms and also

"Unbelievable"

is what our customers say when they see how many computer books we have available. Unlike book "clubs" we have more than just "best sellers" or one publisher's line. We have the most complete selection for both PC's and the Mac. Please call or write today for our latest specials.

COMPUTER BOOKS
PO Box 70746
Sunnyvale, CA 94086 USA
(408) 730-9477 Fridays 11-2 PM PST

Circle **340** on Reader Service Card

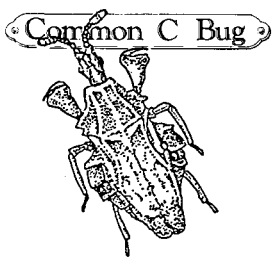
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

T-SHIRTS

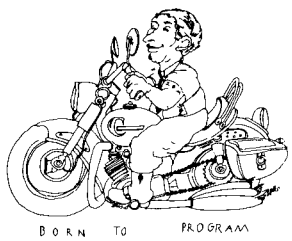
$14.95 each — includes U.S. shipping

Multi-color imprints on the highest quality 100% cotton T-shirts.

SIZES: S, M, L, XL, XXL



COMMON C BUG



BORN TO PROGRAM

PARALLEL PORTS

RS-232 HEAD

BUGS RULE 4 color imprint on a white shirt

PJ ISSUE SPECIAL!

VOL. 6 $24.95*
Complete Set
REG $42.00

VOL. 6

ALL 6 ISSUES OF 1988

1988 started with our 5th Year Anniversary issue and went on to include some of our most requested and referenced issues. Topics include LIM 4.0 specification, OS/2 memory management, 386 and object-oriented programming, VGA graphics, windows, numerics and more. Complete your Programmer's Journal library for 1988 today. Offer Expires 8-31-90.

** Please add $15 for Can/Mex shipping and $30 for all other foreign shipping.*

BACK ISSUES

$7.00 each PREPAID ONLY

A few of the Back Issues currently available

Vol. 6.1: 5th Year Anniversary–LANGUAGES

Vol. 6.2: 386 PROGRAMMING

Vol. 6.3: MEMORY

Vol. 6.4: WINDOWS

Vol. 6.5: OBJECT ORIENTED

Vol. 6.6: NUMERICS

Vol. 7.1: GRAPHICS: High-resolution VGA color modes; graphics processors; Turbo Pascal 4.0 and 5.0 compiled EXE code; Pascal overlay managers; JPI's TopSpeed Modula-2; faster QuickBASIC code; Microsoft's CodeView debugger; COBOL-85 ANSI standard.

Vol. 7.2: OPERATING SYSTEMS: Undocumented DOS features; fast 80386 block moves under DOS; OS/2 dynamic link libraries; OS/2 multitasking; debugging utility for PM and Windows; EGA and VGA animation; disk formatting in Pascal; reviews of windowing and graphics tools, disassemblers, and *The 80386 Book*.

Vol. 7.4: COMMUNICATIONS: A simple terminal program in PM; communications and protocols; database normalization theory and practice; Turbo Pascal 5.5 development tools; Lint utilities for the PC; a millisecond timer for BASIC; reviews of RamNet, DOS/16M, PCX *Technical Reference Manual* and Programmer's Toolkit.

Vol. 7.5: OBJECT-ORIENTED PROGRAMMING: Class libraries in C++; from C to objects; improved C++ code generation; Hal Hardenbergh on the RISC/CISC schism; world scaling in Turbo Pascal's BGI graphics primitives; numeric extensions to C; reviews of 386 development tools, mini-Case tools, and Robert Sedgewick's book *Algorithms*.

Vol. 7.6: USER INTERFACES: Windows terminal program; the CURSES C library; two video displays with Turbo Pascal; VGA and EGA split screens; memory models and static variables for assembly language; object-oriented programming to create a file viewer; large memory spaces from DOS on the 80386; QuickBASIC's string-searching algorithm; reviews of user-interface and mini-Case tools; the *System BIOS for IBM PC, XT, AT*.

Vol. 8.1: NETWORKING: Network File System; an overview of NetBIOS on the PC, from *C Programmer's Guide to NetBIOS* by W. David Schwaderer; fast circle drawing on the VGA; dynamic data exchange in Windows; DESQview API features; computing time in C; a menu system with Turbo Pascal 5.5.

Vol. 8.2: GRAPHICS: *Faster circles for the VGA; the TIGA interface* from TI; the EGA palette; 800x600x16 Super-VGA mode programming; dynamic data exchange for communications; manipulating large data; fast file I/O in QuickBASIC; industry news; 1989 article index.

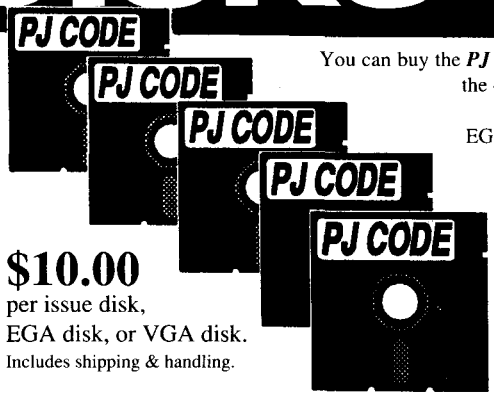
Vol. 8.3: OPERATING ENVIRONMENTS: Undocumented DOS shell; graphics programming in Windows; mousing around: messages in PM; accessing high memory; ASM:advanced segmentation; ellipses on the VGA; up OOP alley in Pascal.

10% discount on any order of 10 or more T-Shirts, Disks or Back Issues.

All mail orders must be accompanied by check or Money Order -- PREPAID ONLY.

CODE DISKS

Use the order form on the card opposite. Order by phone with your VISA/MC. CALL TOLL FREE **800/234-0386** ext 53



$10.00 per issue disk, EGA disk, or VGA disk. Includes shipping & handling.

You can buy the *PJ* code listings from volume 5.5 to the current issue. Code disks for the Michael Abrash "On Graphics" EGA/VGA series are also available.

Diskettes are available for the following issues:

5.5	6.3	7.1	7.5	8.3
5.6	6.4	7.2	7.6	8.4
6.1	6.5	7.3	8.1	EGA
6.2	6.6	7.4	8.2	VGA

360K MS-DOS disks standard; other formats available.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

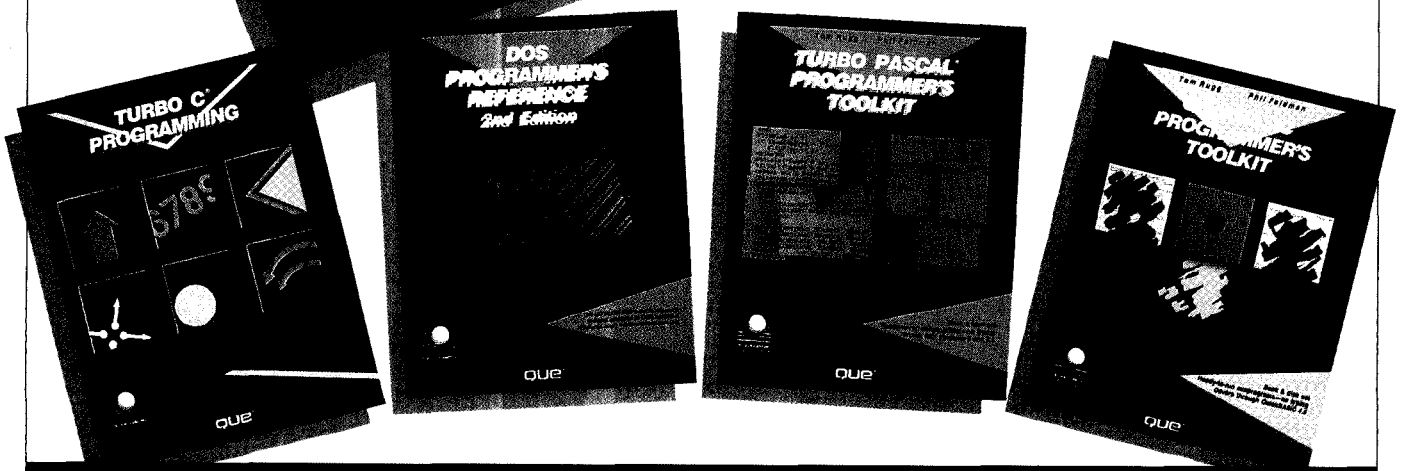
Graphic Results!

Only $24.95

Maximize your EGA/VGA programming graphics with Que's *Power Graphics Programming*! Que has teamed up with *Programmer's Journal* to bring you this nuts-and-bolts graphics programming guide. Now, for the first time, you can have the expertise of Michael Abrash consolidated in a single volume. His insights and approach to efficient, effective programming have earned him high acclaim for his "On Graphics" series. If you are seeking that elusive graphic edge, you can't be without *Power Graphics Programming*!

Order yours directly from *Programmer's Journal*

To order call 800-234-0386. The *Power Graphics Programming* code disk is also available for $14.95. Order your book and code disk today!



Look for these other practical Programming books from Que

Available in better bookstores and computer stores everywhere. Or call 800-428-5331, ext. PJ01.

que®

Circle **296** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DOS & OS/2
Free Demo Diskette Available

Q. How many programmers does it take to maintain a MAKE dependency file?

A. NONE! *If you use* VersiMAKE™

VersiMAKE™ is a full-featured MAKE utility that includes:

- **Dependency Generation**
  Derives your system's dependencies, through analysis of its C and MASM source files. Say goodbye to manual maintenance of MAKE dependency files!
- **Wild Card File Name Matching**
  Analyzes an entire collection of source files with a single statement.
- **Nested Include Files**
  Handles standard C and MASM "include" conventions, and the INCLUDE environment variable.
- **Powerful Macro Facilities**
  Built-in macros, user-defined macros, and environment variables.
- **Analytical Reports**
  Shows the entire Include file hierarchy for each source file analyzed, and all of the parent source files for each Include file.

Q. How many programmers does it take to trace a symbol thru your system?

A. ONE! *If you use* VersiCREF™

VersiCREF™ is a unique utility that creates a Master Cross-Reference of your entire system.

- **Multi-Lingual**
  Handles C, assembler, or both.
- **Flexible**
  File names with line numbers, or file names alone. Global and local symbols, or globals alone.
- **Powerful**
  Easily handles systems containing 100 source files or more.

VersiMAKE™	$125
VersiCREF™	$75
Both	$150

(Outside U.S., add $5 for shipping and handling)

800-344-4096
(In NJ, 609-779-0784)
MC/VISA/AMEX accepted

SUMMIT INFORMATION SYSTEMS, INC.
496 N. Kings HWY Suite 130
Cherry Hill, NJ 08034

comes in a version that can be ported to most C environments. Library source code is included and no royalties are charged. A license for PANEL Plus II costs $495 (upgrades are $75).

Roundhill Computer Systems Ltd.
1964 Richton Drive
Wheaton IL 60187
(708) 690-3737

SILVERCOMM "C" ASYNC

SilverWare has released the SilverComm "C" Async Library, a powerful communications library for C programmers. The SilverComm library of more than 160 routines is designed to provide reliable high-performance, interrupt-driven control of asynchronous communications in C programs.

SilverComm routines have support for huge model queues exceeding 64K, terminal emulation, XMODEM and YMODEM file transfer protocols, and background timers. Low-level library routines can exploit baud rates to 115K, flow control (3 hardware types and XON/XOFF protocol), the Smartmodem command set, high-level remote input, character filtering, and video functions. SilverComm routines incorporate a unique Device Event monitor capability that permits execution of C functions at interrupt time. Functions can set flags signaling the application of an important event. The low-level assembly language routines are fully interrupt-driven and use a table design for maximum flexibility.

SilverComm supports IBM's Dual Async adapter (16550 UARTs), Digiboard 4-,8- and 16-port boards, the AST-4 port adapters, and an unlimited number of COM ports. It also supports the Norton Guide database. Full source code is included. SilverComm is priced at $249.

SilverWare, Inc.
3010 LBJ Freeway, Suite 740
Dallas, TX 75234
(214) 247-0131

DATA JUNCTION 3.01

Tools & Techniques has released version 3.01 of Data Junction, its unique data conversion tool. With its powerful "Hub & Spoke" architecture, users can translate native data files to and from other file formats. Data Junction provides a timely solution to the problems of data incompatibility. Records, fields, and bytes may be sorted, extracted, rearranged, edited, and entered into the exact output format required. Parameters may either be selected interactively or set up for automatic batch mode operation. DOS licenses are $99 for the Standard version of Data Junction, $199 for the Professional version, and $299 for the Advanced version.

Tools & Techniques Inc.
1620 West 12th Street
Austin, TX 78703
(800) 444-1945

NEW WATCOM COMPILERS

Watcom has announced new versions of both their C compiler (C8.0) and FORTRAN F77 development systems for DOS, OS/2, and 386 DOS extenders. Both the C and FORTRAN development systems include compilers that offer extensive code optimization along with support tools—a windowed source-level debugger, linker, librarian, and profiler. The code generated by F77 is run-time compatible with WATCOM C, allowing inter-language function calls. 386 versions of both the C and FORTRAN compilers and tools support the development of programs which run in the 32-bit protected-mode, using the Phar Lap and Ergo 386 DOS extenders.

Language support has been extended to include IBM's SAA C specification as well as the ANSI C standard and Microsoft C source compatibility. The Express C integrated environment is included with the C compilers. Standard and professional editions of C8.0 and C8.0/386 will be available. The professional editions provide extra features such as OS/2 and Windows support, an execution profiler, a graphics library, and a 386 protected-mode compiler. The professional edition of C8.0/386 provides the 386 source-level debugger.

The standard C8.0 costs $395; the professional edition, $495. C8.0/386 is $895, with the professional edition $1295. The price for F77 V8.0 is $495 ($1095 for F77/386 V8.0).

Watcom Products Inc.
415 Phillip Street
Waterloo, Ontario
Canada, N2L 3X2
(519) 886-3700 PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

386 DOS Extenders

By M. Steven Baker

When 386 DOS extenders were first introduced, many industry watchers predicted their early demise. And now, a few years later, 386 DOS extenders are solidly entrenched in the PC developer's market. MS-DOS remains the operating system standard for the IBM microcomputer world and the number of high-end commercial applications built with 386 DOS extenders continues to grow.

Several 386 DOS extenders are now available (see Table 1). Extenders from Phar Lap Software and Ergo Computing (formerly Eclipse Computer Solutions and A.I. Architects) represent the bulk of the market. The IGC DOS Extender is being marketed to OEMs and hasn't been updated for over 18 months. A newer entry developed at the University of Salford (OTG Systems) works only with their FTN77/386 system (see article on Page 53). MicroWay is bundling a version of OS/386 with the latest version of their compilers. This product emulates Phar Lap and is for development purposes only.

All of the DOS extenders emulate DOS and the BIOS to a substantial degree (registers are widened to 32 bits). Since both Ergo and Phar Lap DOS extenders are available for distribution and span development tools, I will cover these products in more detail.

ERGO OS/386

Ergo Computing markets two DOS extenders: OS/386 designed for 386 machines and their Hummingboard 80386/80387 coprocessor boards and OS/286, a similar DOS extender designed for 80286 machines. The Ergo Developer's Kit includes the OS/386 kernel, a symbolic debugger, and several utility programs. Version 2.0 includes an automatic configuration program along with support for Virtual Control Program Interface (VCPI).

The Ergo OS/386 runtime kernel differs substantially from the other vendors' products. OS/286 supports a multiple segmented memory model on the 286, allowing applications to break the 640K memory barrier under DOS. The same memory map is supported on the OS/386 runtime along with the more common flat, unsegmented (small) memory model. Phar Lap hardwired selectors are also supported.

For development purposes, the kernel is installed as a TSR (which can be removed from memory when not needed). This speeds the loading of protected mode programs during development and testing. The kernel runs at ring 0 (most privileged) while your protected mode program runs at ring 3 (least privileged). The I/O privilege level (IOPL) for applications is set at 3 so that your application can do INput and OUTput to hardware ports directly. The resident OS/386 kernel can be configured to use only about 60K of the lower 640K DOS memory space.

OS/386 switches from protected mode to real mode for DOS calls and uses paging. Memory both under and above the 1-megabyte barrier is available for use in the protected mode. The Weitek math chip is supported using the Phar Lap convention (segment FS:0 points to the Weitek chip). Extended DOS calls are supported for calling real mode procedures, handling interrupt vectors, creating code and data segments, getting segment information, and making block transfers to low memory. Extended DOS calls are made with **AH > E0H** using INT 21H. During development, execution of a 386 program is preceded by the UP command.

OS/386 supports a powerful Real Procedure Call (RPC) mechanism that facilitates communicating with real-mode routines such as graphics and communications located in low memory. RPC's allow 386 applications to use real mode libraries until 386 libraries are available. The OS/386 RPC mechanism allows RPC's to be written in either C or assembly language.

Currently, the 80386 compilers that work under the OS/386 runtime and produce 32-bit code are MetaWare's C and Pascal, Watcom C/386, and Lahey FORTRAN F77/L32. The Lahey 32-bit linker and librarian are supplied by Ergo for development. A Virtual Memory option is available at an additional cost. Among the three runtime environments, OS/386 offers the closest emulation of DOS and BIOS.

PHAR LAP 386|DOS EXTENDER

Phar Lap markets a 386|DOS Extender (RUN386), a 386 assembler, linker, librarian, and debugger package (386|ASM), and a separate symbolic debugger (386|DEBUG for $295). Phar Lap has been the traditional supplier of the assembler and linker used for 386 DOS development. Phar Lap defined the 386 object module (EASY-OMF) most commonly used under DOS for the 386. Most of the 386 compilers generate Phar Lap 386 object files and are supplied with Phar Lap-compatible libraries.

Under development, a 386 application is invoked by prefixing the command line with RUN386. Phar Lap supports two different EXP file formats: the original EXecutable Protected mode (EXP) and a P3 format EXP that supports a packed mode that creates smaller executable file sizes.

The Phar Lap runtime is a flat memory model with several hardwired segment selectors for memory mapping. Both RUN386 and your application run at ring 0 (most privileged). This allows complete machine control from within your application. RUN386 supports both enabling or disabling paging. With paging, RUN386 can also use memory below 640K for 386 applications. The Phar Lap DOS extender uses approximately 100K of low memory, leaving about 500K of low memory plus any extended memory above 1 megabyte available for protected mode use.

The DOS protected mode function calls are similar to normal DOS calls with a few exceptions. DOS extender calls use **AH=25h** with the subfunction in **AL**. The Weitek math chip is supported by mapping it into a 64K block of memory pointed to by segment

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TABLE 1: 80386 DOS EXTENDERS

	Ergo	PharLap	IGC	OTG
Product	OS/386	386|DOS-Extender	X-AM (1)	DBOS
Version	2.1.01	2.2d	1.2	2.2
Developer Price	$495	$495 (2)	$595 (1)	(4)
Unlimited Runtime	$300 (3)	$1,995 (2)	-0- (1)	(4)
Virtual Memory	$795	$295	none	included
VM Runtime	included	$1500 to 10,000	none	included

(1) X-AM is now primarily marketed to OEM customers, and may no longer be available for retail sales. The X-AM Developer's Kit includes an unlimited runtime license. IGC, 4800 Great America Parkway, Santa Clara, CA 95059; (408) 986-1431.
(2) The Phar Lap Developer's Kit includes their 386 assembler, linker, librarian, and mini-debugger as well as RUN386 (386|DOS Extender). The unlimited runtime costs $1,995 per product for the first 1,000 copies. Additional copies are priced at a percentage of the retail price of the application.
(3) Ergo licenses under 20 copies of a single executable for a cost of $100. The cost is $300 for 20 or more copies.
(4) OTG Systems (see article on page 53) markets a 386 DOS extender, but this product is currently supported only by their FORTRAN FTN77/386 development system. The runtime costs $50/copy in small quantities.

register FS. The majority of 386 compilers generate code to operate under Phar Lap's DOS extender. These compilers include those by MetaWare, LPI, SVS, MicroWay, Alsys, and Watcom.

RUN386 supports the VCPI jointly drafted by Phar Lap and Quarterdeck (DESQview) to allow multiple 386 programs to cooperate. Phar Lap was the first vendor to support virtual memory and a number of commercial products already incorporate the Phar Lap Virtual Memory Manager (VMM). This DOS extender makes the largest quantity of memory available for 386 programs. This can be a consideration for machines with relatively low memory (2 megabytes).

Ergo Computing
One Intercontinental Way
Peabody, MA 01960
(508) 535-7510

Phar Lap Software
60 Aberdeen Avenue
Cambridge, MA 02138
(617) 661-1510

Instant Memory...Instant Savings!

Save $50—A Special Offer For the SilverWare "C" EMM Library.

The SilverWare "C" EMM Library is an interface to the Expanded Memory Manager (EMM) device driver. It's designed to relieve the "memory crunch" caused by the limitations of DOS.

The "C" EMM Library contains more than 60 functions that provide a consistent, comprehensive link between your "C" application and the EMM Driver.

Using the "C" EMM Library, you can add up to 32 megabytes of extra space for data storage, instead of stripping down your application to fit in the space DOS allows.

Memory Model Flexibility

The SilverWare "C" EMM Library gives you the freedom to operate correctly in any memory model. It provides memory model-independent functions that access far memory objects in small and medium memory models.

In addition, the "C" EMM Library includes functions that call multiple functions in EMM—so you can access EMM's power with just a few keystrokes. You don't have to waste time translating Expanded Memory Specifications or writing assembly language interface functions!

All This—And Memory Too

The SilverWare "C" EMM Library gives you much more than just expanded memory:

- Support for Microsoft C and Quick C, plus Borland Turbo C
- Thoroughly commented source code for easy modification
- Royalty-free distribution
- Complete, easy-to-read documentation with examples for each function
- The Norton Guides Database on-line Help
- Free technical support

Don't Forget—Order Now!

The SilverWare "C" EMM Library normally sells for $199. But for a limited time, you can get it at the special introductory price of just $149.

And like all SilverWare products, the "C" EMM Library is backed by an unconditional money-back guarantee. If you're not completely satisfied, just return it for a full refund.

Order your copy of the SilverWare "C" EMM Library now by calling (214) 247-0131. But hurry—this offer ends soon!

You've got nothing to lose—but your memory!

Silver WARE Inc.
3010 LBJ Freeway
Suite 740
Dallas, TX 75234
(214) 247-0131

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

FIGHT PIRACY & PROTECT YOUR PROGRAM $$$'s!

Since 1986, companies worldwide have been choosing Az-Tech security products. If you demand the strongest protection available, why not choose one of these "proven leaders":

- **EVERLOCK** Copy Protection
- **EVERTRAK** Software Security
- **EVERKEY** Hardware "Key" Software Security

For IBM and Compatibles.
30 day money back guarantee.
Free info and demo disk available.

Az-Tech Software, Inc.
305 East Franklin
Richmond, MO 64085
(800) 227-0644 (816) 776-2700
FAX (816) 776-8398

Circle **51** on Reader Service Card

HUGE Professional Shareware Libraries

Comprehensive, indexed collections of all known Public Domain and Shareware utilities for PC professionals, updated quarterly with the latest products. Includes extensive disk based or printed directories with multiple search options. Each library designed specifically with the PC professional in mind. Computer Language called our dBase product *"possibly the most productive $100 you'll ever spend"*. VISA, MC, AmEx,POs, COD.

Product	Disks/Files	Price
dBase/dBase Compilers	56+/1100+	$99.50
AutoCAD	17+/570+	$99.50
C	19+/400+	$49.50
Turbo Pascal	23+/270+	$49.50
Ventura DTP	21+/170+	$49.50
Lotus 123	17+/200+	$49.50
Assembler	10+/170+	$49.50
Games	35+/220+	$49.50
Novell Netware	17+/170+	$49.50
WordPerfect	30+/160+	$49.50
Utilities for DOS Gurus		$49.50

EMS, 4505 Buckhurst Ct.
Olney, MD 20832 USA
(301) 924-3594 Fax: (301) 963-2708
France (1)47763872 Germany 0911/338844
Benelux 31-20-5753019

Circle **353** on Reader Service Card

EZ-COPY PLUS

Flawless Disks Fast!

•2X+ faster than DOS •Read diskette once, then..•Quickly & accurately mass duplicate 5.25" & 3.5" disks (360K, 720K, 1.2M, 1.44M) on your own PC/XT/AT/etc. © •Supports up to 4 drives with automatic start and switching. •Formats, copies, verifies, etc. • Partial copies too.•Even faster with pre-formatted disks. •Serializes disks and/or labels, in 1 smooth operation. •Save disk images to HD, more... •Replaces hardware worth $1000s.

Only $129 +s/h

V/MC/AX Orders & Brochures Toll Free:
1•800•359•9539 xPJ790
INFO 1-713-280-9900

S&H: $6 USA-48; $10 N.Amer; $20 Foreign.
EZX Publ., 917 Oakgrove Dr., Ste PJ790, Houston, TX 77058 •FAX: 1-713-280-0525

Circle **352** on Reader Service Card

FREE CATALOG!

LATEST AWARD BIOS
PC/XT, 286, 386

COMPILED BASIC TOOLS
Libraries, Utilities
for QuickBASIC 4.5

HARDWARE UPGRADES
for PCs & PC/XTs

(800) 423-3400 - (412) 782-0384
KOMPUTERWERK INC.
851 Parkview Blvd.
Pittsburgh, PA 15215

Circle **155** on Reader Service Card

RELATIONAL DATABASE LIBRARY

For TurboC, Microsoft C on DOS systems. The 114 functions in **"c_wndw"** and **"c_ndx"** libraries support:

- Multiple B-tree index files
- Stacked, overlapping windows
- dBase .DBF and .NDX file formats
- Multi-level file relationships
- Pull-down and pop-up menus
- Hashed random file access
- Data dictionary field access

Libraries $99.50; Source $199.50; VISA/MC

Marietta Systems, Inc.
P.O. Box 71506
Marietta, GA 30007
(404) 565-1560

Circle **223** on Reader Service Card

TLIB™ is FASTEST!

Times are to update a 45K library on a PC/XT. PVCS and TLIB 3.0 are from Sept 87 *PC Tech Journal*. MKS RCS 4.2 and TLIB 5.0 are newer.

TLIB™ is *BEST!*

"Do not be fooled by the fact that this is the least expensive of the five packages reviewed here - TLIB has features and power to spare" John Rex, **Computer Language**

"TLIB is a great system" J. Vallino, **PC Tech J**

- **Full-Featured Version Control for Software Professionals.** Check-in/out locking. Branching. Keywords. Wildcard and list-of-file support. Can merge parallel changes and undo intermediate revisions. Network and WORM support. **Mainframe compatible** deltas for Pansophic, ADR, IBM, etc.. integrates with **Opus™ MAKE & Slick™ MAKE.**

MS-DOS $139, OS/2 $195 + shipping Visa/MC
5 station LAN license $419 (OS/2 $595), call for other sizes

BURTON SYSTEMS SOFTWARE
PO Box 4156, Cary, NC 27519 **(919) 233-8128**

Circle **288** on Reader Service Card

COBOL *Magic*™

The exciting new tool for source code analysis

COBOL Magic makes it easy to find your way around the largest, most complex programs. Without typing, you can instantly locate anything you need to see. **COBOL Magic** prevents you from getting lost while following a long series of PERFORMs or GOTOs. COPY libraries can be imported into the text.

If you're still using text editors, printed listings, cross-references, or diagrammers to analyze your COBOL source, discover the power of **COBOL Magic!**

FREE 30-DAY TRIAL

Jacobson Software Inc.
4221 Hemlock Lane
Plymouth, Mn. 55441
(612)550-1609

Circle **28** on Reader Service Card

FREE TRIAL

Pinnacle File Manager

We guarantee immediate productivity gains from using our C file manager library. Simple table-and-column view of data with advanced B-Tree indexing and hi-speed cache frees your brain to solve application-specific problems.

Vermont Database Corp.
RR3 Box 6516
Stowe, VT 05672
802-253-4437

Circle **328** on Reader Service Card

---

mxw Windowing System

ONE library compatible with MS Assembler, BASIC, C, FORTRAN, Pascal, Quick BASIC, and Quick C.

Not multiple library versions, but ONE LIBRARY that links to all of the above language systems.

- Multiple window designs
- Powerful screen management
- BIOS/DOS interface
- Includes multi-facet menuing system
- Build top screen menuing systems with pull-downs, etc.
- Exciting text color and format control features (in-out of windows)

$59.95

Call for FREE Day Calendaring System that presents mxw features.

(516) 434-1979

Send Check/MO to:
MTD SOFTWARE
21 Ralph Ave, N. Brentwood, NY 11717

Circle **312** on Reader Service Card

---

dBASE FILE ACCESS FROM C

Code Base 4 is a library of C routines which gives complete dBASE and Clipper functionality and file compatibility. The routines correspond directly to dBASE commands. Create multi-user applications using pulldown menus, popup windows and data entry.

Code Base 4 supports DOS, UNIX, OS/2 and Microsoft Windows. $295 with Source! FREE DEMO.

Sequiter Software Inc.
P.O. Box 5659, Station L
Edmonton, Alberta Canada T6C 4G1

403-448-0313

Fax: 403-448-0315

Circle **269** on Reader Service Card

The $25 Network

Try the 1st truly low cost LAN

- Connect 2 or 3 PCs, XTs, ATs, PS/2s
- Uses serial ports and 5 wire cable
- Runs at 115K baud, up to 90 feet
- Transfer 8500 bytes per second (ATs)
- Runs in background, totally transparent
- Share any device, any file, any time
- Needs only 14K of ram
- Replace all file transfer software
- $25 per network, NOT PER NODE!
- OVER 15,000 SOLD WORLD WIDE

Skeptical? We make believers!

iModes Information Modes
Drawer F, Denton, TX 76202
817-387-3339 Technical
1-800-628-7992 Orders

Circle **348** on Reader Service Card

OPT-TECH SORT/MERGE

Extremely fast Sort / Merge / Select utility. Run as an MS-DOS command or CALL as a subroutine.
Supports most languages and filetypes including Btrieve and dBase. Unlimited filesizes, multiple keys and much more!

MS-DOS **$149.**
OS/2, UNIX **$249.**

Opt-Tech Data Processing
P. O. Box 678
Zephyr Cove, NV 89448
(702) 588-3737

Circle **16** on Reader Service Card

Offer your product directly to Professional Programmers:

Placing an ad in **PJ BILLBOARD** is an easy, low-cost way to reach potential buyers. Regular listings include a headline, company information, and approximately 400 characters for your message. Display ads are 1 5/8" wide by 2 3/8" high. For more information or to reserve a space, call our ad department at **503-747-0800.**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DO IT YOURSELF!

XenoCopy-PC

PC-DOS program lets your PC
Read/Write/Format
over 350 formats **$79.95**

XenoFont
high quality text screen printouts
$49.95

+ $5.00 S/H + Tax if CA.
XenoSoft • (415) 644-9366
2210 Sixth Street,
Berkeley, CA 94710

Circle **123** on Reader Service Card

sPORTt
serial PORT tester

sPORTt is a serial port tester for people who don't want to be data communications experts, but use or write software for serial devices, such as plotters, mice, digitizers, modems, serial printers, or computer to computer links.

- displays all data going in and out in ASCII, decimal, hex, binary
- lets you choose handshaking lines
- sends/receives disk files
- automatically runs port diagnostics

Order today only $89.95!

WHS
3037 Grass Valley Hwy #8201
Auburn, CA 95603
916-885-2480

Circle **329** on Reader Service Card

PC TIMER TOOLS
New Version!

PCHRT is the definitive answer to execution profiling, precision timing and timer tick interrupt management in a PC/MSDOS environment. Over 50 functions manage multiple microsecond timers, generate precision delays, insert timers in most any interrupt, and run the timer tick interrupt at a user specified rate. Utilities to time execution and profile interrupts of any .EXE. Supports TC, TP, and MSC. Full library source included. $49.95 ppd USA, elsewhere add $4.00. VISA/MC.

RYLE DESIGN
PO Box 22, Mt. Pleasant MI 48804
517-773-0587

Circle **210** on Reader Service Card

IS 640K JUST NOT ENOUGH?

The ATLAS MEMORY BOARD adds 64K of DOS USABLE memory to the base 640K, making the total CONVENTIONAL memory 720,896 bytes. It does not use any memory resident drivers, and is compatible with all known applications. The same board works in any PC, XT, AT or 386 compatible, with monochrome graphics or CGA video. Memory added is NOT extended or expanded memory. The Atlas memory board has been certified by both NOVELL and 10 Net. Most orders shipped same day.

ATLAS MEMORY BOARD
15 Day Free Evaluation **$149.00** *Qty 1 Quantity Discounts Available*

M/C **1-800-535-6712** VISA
9AM - 9PM MON-SAT
NATIONAL DISCOUNT COMPUTERS
702-459-0049 NV Residents & Tech Support

Circle **107** on Reader Service Card

Why you want BATCOM!

BATCOM is a batch file compiler that compiles your ".bat" files to ".exe" files to make them faster, more professional, and more capable. BATCOM extends DOS with new commands so you can read keyboard input, perform arithmetic, use 'subroutines, and much more. In addition, BATCOM protects your source code, and you can distribute your compiled programs without royalties. Only $59.95

ORDER TODAY!

Wenham Software Co.
5 Burley St.
Wenham, Ma. 01984
(508) 774-7036

Cirlce **320** on Reader Service Card

QuickBASIC File Indexing

Creating B+ Tree indexes in QuickBASIC with **Index Manager™**. Access files randomly by full or partial key, or sorted forward or backward. One module performs all functions making **IM** very easy to use. Only indexes managed! You retain full control over your data. Assembler code and cache buffers make **IM** very fast. Demo on GEnie and CompuServe. Eight sample programs and 64 page Users Guide included.
Only $59 (+tax in CA). VISA/MC accepted.

CDP Consultants
1700 Circo Del Cielo Drive
El Cajon, CA 92020
(619) 440-6482

Circle **335** on Reader Service Card

Developer's Assistant

A comprehensive system providing the essentials all developers need. Database to maintain users and dealers. Uses multiple indexes for fast lookups. Lots of reports. Utilities to embed serial numbers in your program's sign-on screen. A label designer and more. Professional directory of packagers, duplicators, associations, lawyers, etc. Directory of Editors for Product Releases.

Plus **FREE** *use of a 900 number to* support your product and earn money by the minute. $69.00 (+tax in CA)

S.L.A.M. Software
2523 El Portal Dr. Suite 101-25
San Pablo, CA 94806
415-237-9514

Circle **12** on Reader Service Card

TURBO PASCAL PROGRAMMERS

TP3, an easy-to-use preprocessor, permits usage of assembly language rather *than in-line statements to generate in-line* machine code. The assembly source is retained as commentary.

All 8086 and 8087 instructions are supported using MASM syntax. Pascal source passes through unchanged.

A really useful addition to your utility library. 5.25" ds/dd (xt) floppy. $49

MAIKON SOFTWARE
8304 Chianti Court
San Jose, CA 95135
(408) 223-1166

Circle **345** on Reader Service Card

The EDitor

Powerful **UNIX**-style programmer's editor. For use on IBM or compatible. Comprehensive manual included. A solid value at only $74.95 (US).

Available *from:*

Blue Jay Systems
PO Box 608
Mount Vision, NY 13810
607 293-8007

Circle **11** on Reader Service Card

GRAF/DRIVE PLUS

Turbo C/Pascal Graphics Drivers for Printers and Plotters

LaserJet–Epson–HP7470. Just plug our BGI-compatible drivers into Turbo's *initgraph.*

- Port your screen graphics program to hard copy devices with only minor changes.
- Get the full resolution of the output device, not a screen dump.
- Not TSR, Max 16K.

Personal $149. No-royalty developer $299. 30-day m/b guarantee.

FLEMING SOFTWARE
PO Box 528 Oakton, VA 22124
(703) 591-6451

Circle **178** on Reader Service Card

SHAREWARE AUTHORS

Give your programs the chance they deserve! With Megapost, your program is uploaded to CompuServe, GEnie, Delphi, BIX, and EXEC-PC. The low fee *is probably less than you would pay just* to open accounts with all of these services. Included are one half-price update and notification of the downloading procedure and 3-month download count for each service. 15% discount for ASP members.

Write for a free brochure.

Andrew M. Saucci, Jr.
641 Koelbel Ct.
Baldwin, NY 11510-3915

Circle **347** on Reader Service Card

80386 MACRO DISASSEMBLER

MASTERFUL DISASSEMBLER is the worlds first macro disassembler. Interactive by design not after thought. Features:

- User defined macros with arguments
- Auto code/data separation
- Auto commenting and label naming
- Command menus and help screens
- Customizable disassembly options
- Comprehensive, indexed, 100 pg manual
- Fully supports 8086/87 thru 80386/87
- COM/EXE/SYS or other files or memory
- Cross reference and much much more!

Still only $67.50 + tax ($1.50 s/h).

C.C.SOFTWARE
1907 ALVARADO AVE.
WALNUT CREEK, CA 94596
(415)939-8153

Circle **132** on Reader Service Card

Coming Next Issue:
Object-Oriented Languages

September/October 1990 Vol. 8.5
Advertising Space Reservation:
July 6, 1990
Materials Due:
July 13, 1990

Please call *Programmer's Journal* to reserve your space.
503-747-0800

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ADVERTISER INDEX

Reader Service	Advertiser Name	Page Number
305	Acme Software	64
331	Addison-Wesley Publishing Co.	45
345	Alphax Systems, Inc.	51
41	Applied Intuition	65
30	Austin Code Works, The	63
207	Automation Education	64
51	Az-Tech Software, Inc.	93
14	Blaise Computing, Inc.	15
2	Blaise Computing, Inc.	17
15	Blaise Computing, Inc.	19
11	Blue Jay Systems	94
3	Borland International	C4
288	Burton Systems Software	93
132	C. C. Software	94
321	Catenary Systems	28
335	CDP Consultants	94
281	Cobra Systems Software	85
39	Coleman Softlabs	5
340	Computer Books	87
229	Disk Software, Inc.	30
31	Double R Software	22
353	EMS	93
352	EZX	93
178	Flemming Software	94
261	Genus Microprogramming	51
*	Gimpel Software	66
18	Goin Places Software	70
334	Hamilton Laboratories	52
42	Harvard Softworks	85
348	Information Modes	93
358	Inspired Innovations	10
28	Jacobsen Software	93
211	Kadak Products, Ltd.	79
295	Knowledge Dynamics Corporation	44
155	Komputerwerk, Inc.	93
63	Laboratory Microsystems, Inc.	80
44	Lahey Computer Systems, Inc.	58
354	Maikon	94
223	Marietta Systems, Inc.	93
*	Microwork	7
272	Midnight Engineering	81
240	Milton Brown Software	62
121	MISOSYS, Inc.	32
266	Mix Software	69
94	MMC AD Systems	65
312	MTD Software Development Co., Ltd.	93
109	Nanosoft	51
107	National Discount Computers	94
160	Nu-Mega Technologies	3
23	OOP Directory	C3
16	Opt-Tech Data Processing	93
270	OTG Systems, Inc.	57
*	Paradigm Systems	40
208	PC Tech, Inc.	30
36	PC Tech, Inc.	80
6	Periscope Company, Inc., The	26
*	***PJ* BBS Listing**	73
*	***PJ* Code DiskScription**	8
279	Pocket Soft, Inc.	9
*	***Programmer's Journal* Mail Order Store**	88
252	Quantasm Corporation	81
147	Quarterdeck Office Systems	C2
296	Que Corporation	89
19	Quintus Computer Systems, Inc.	40
34	R Fringe Publishers, Inc.	48
20	Rational Systems, Inc.	7
210	Ryle Design	94
12	S.L.A.M. Software	94
26	Sage Software, Inc.	12–13
32	Sandpaper Software	14
347	Saucci, Andrew M., Jr.	94
269	Sequiter Software, Inc.	93
7	Silicon Valley Software	56
43	Silverware, Inc.	92
165	SLR Systems	78
98	Software Engineering Store, The	23
110	Softway, Inc.	31
47	Solution Systems	16
136	Sophco	1
330	Stanwood Associates, Inc.	27
158	Summit Info Systems, Inc.	90
289	TechnoJock Software, Inc.	41
166	Tool Makers, The	71
327	Tools & Techniques, Inc.	44
276	Tri-Technology Systems, Inc.	65
25	TurboPower Software	47
309	UnderWare, Inc.	50
341	V Communications, Inc.	68
125	V Communications, Inc.	77
29	Venn	73
328	Vermont Database Corporation	93
304	Visitech Software	28
*	Watcom Products, Inc.	59
320	Wenham Software Co.	94
100	Western Wares	62
329	WHS	94
123	XenoSoft	94
182	Zortech, Inc.	54

When contacting any of the above advertisers, please tell them you saw their ad in ***Programmer's Journal*** and that you appreciate their continuing support of this magazine. The advertisers marked with an asterisk (*) prefer to be contacted directly. ***PJ***'s Advertiser Index is provided as a free service, and ***Programmer's Journal*** cannot be held responsible for omissions or errors. ***Programmer's Journal*** is not responsible for errors in advertisements.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

OFFRAMP

By Hal W. Hardenbergh

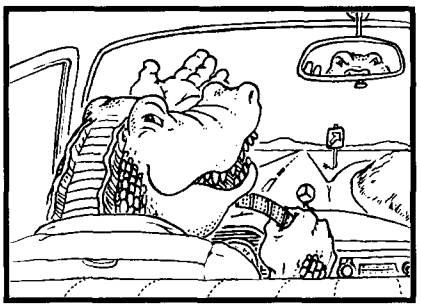


ILLUSTRATION BY JIM CARPENTER

Real computer periodicals have inside-the-back-page columns. Now *Programmer's Journal* is apparently trying to join the Establishment. Unfortunately, somebody has to write the column. The job calls for a writer with the IQ of a dead sea slug, the moral posture of a tenement landlord, the calm detachment of a school of piranhas in a feeding frenzy, and a little knowledge of computers and programming. Liz, Steve, and Colin quickly identified a fully qualified individual. Regrettably, John Dvorak was unavailable.

If you've been reading the business section of your local newspaper lately, you've seen a lot of conspiracy theories as to why the Japanese are so (unfairly!) successful at making and selling DRAMs. All of these theories place the Japanese at fault.

Differences in the way industrialized countries perform are almost always based on cultural or structural differences. For example, in Sweden or England nobody ever goes hungry, whether they work or not. As a result, Sweden's economy is in trouble, and England is on the verge of becoming a third-world nation. It's a regrettable fact that, if people don't have to work, lots of them just won't.

In this country and in Japan, it's possible to go hungry if you don't work. The flip side of this is that you can make, and keep, a substantial amount of money in the U.S. and Japan if you're successful. Witness Donald Trump, Bill Gates, and Steve Jobs. Sweden's and England's tax system is essentially confiscatory above a certain level, which is how they feed all those folks who don't work. It's a dirty shame, but it seems that to be permitted to succeed, you also have to be permitted to fail.

Gorbachev's reforms are in trouble in Mother Russia because the people are not sure they want the privilege of failing—something that comes automatically with a capitalistic economic system. Hungary and Poland, on the other hand, are looking forward to capitalism because that's how things worked within the memory of many of its citizens. (Yes, I subscribe to the British magazine *The Economist*.)

Some folks believe that Japan is successful in making VCRs because their best engineers are designing their VCRs, while our best engineers are designing B-2s, whose sole apparent mission is to bomb poppy fields in Colombia and Peru.

A more important reason is that the Japanese adopted the American Deming's quality control ideas a few decades back. Deming's ideas involve quality circles and cooperation among equals, even if some of those equals are managers and others are workers. That works in Japan, whose society is based on consensus. American companies are trying to catch up, but they can't.

American society is based on "I am the boss, you are less than nothing, and you will do what I say!" Look at the way Detroit treated its production line workers in the '60s, or the way Ashton-Tate's Ed Esber treated his programmers the last few years. Quality circles cannot succeed in such a social environment, which is why American consumers correctly believe that Japanese cars are more reliable than Detroit's.

On the other hand, a take-charge attitude is what it takes to design a first-rank microprocessor. Japan has been unable to produce even one such design. Hmm. I wonder if Japanese newspaper business sections publish conspiracy theories over the U.S.-Japan microprocessor gap?

Why are capital costs (the cost of business loans) in this country double those in Japan? The interest income from a savings deposit in this country is taxable. In Japan it is not. Each is due to the laws of the respective countries, laws which the legislatures of either country can change at any time.

The U.S. taxes interest on savings to deliberately discourage savings. The U.S. wants its consumers to spend, not save. When consumers spend there's not likely to be a recession. Japan, on the other hand, doesn't tax interest income because the country wants a high rate of savings so that there will be a large capital pool to finance Japanese industrial expansion. If there's lots of savings available, then the capital is there for industrial expansion, and borrowing costs will end up lower because of simple supply and demand.

(Please don't dismiss the idea that countries deliberately use tax policy to set social goals. For example, mortgage interest is tax-deductible and rent isn't because our government deliberately wants to encourage home ownership. A homeowner is a better citizen because he or she has a stake in the community. And the government always knows where to find a homeowner...)

Why are Americans obsessed with corporate performance at the end of the next quarter, while the Japanese take a longer view? In Japan, banks commonly own large chunks of the companies that borrow from them. Banks, by their nature, take the long view. American banks also take the long view, but it's illegal for an American bank to own part of a company! Again, Congress could change this overnight if it wished.

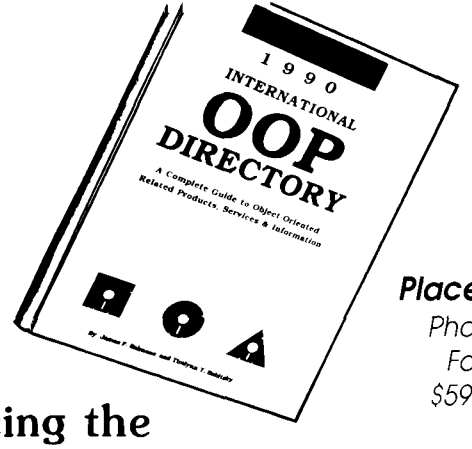
The first round of 64K DRAMs were highly unreliable, no matter who made them. The U.S. operates under the restrictions of the ancient Sherman Anti-Trust Act, which forbids most types of cooperation among U.S. companies. Japan has no anti-trust law. So MITI stuck the best DRAM designers from five Japanese companies under the same roof to design the second generation of Japanese 64K DRAM. In this country, that would have been highly illegal. Intel, on its own, was unable to develop a reliable 64K DRAM and dropped out of the market along with some other U.S. vendors. Intel *invented* the DRAM! Congress can repeal the Sherman Act any time it wants.

To make DRAMs you need lots of capital to build enormously expensive fabs which take 18 to 24 months to bring on line, so you aren't going to be making money for a long time. And unless you build quality parts, nobody will buy your DRAM.

Gee. I wonder why it is that the Japanese make and sell more DRAMs than we do? Can any of you readers think of a reason for their success?

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Go Directly to the Source



Place Your Order Now
Phone: (212)274-0640
Fax: (212)274-0646
$59 US — $69 Foreign

Limited First Printing

Introducing the 1990 International OOP Directory

The complete source for OOP-related information.

Unlike any other buyer's guide on the market today, this unique publication covers virtually every aspect of the O-O field. That means you'll find detailed information on companies, products, consultants, reviews, significant published articles, books, training, conferences and inside info — in one convenient sourcebook ... **The 1990 International OOP Directory.**

The facts you locate will be accurate, complete and up-to-the-minute — providing you with reliable information needed before you make an OOP-related purchasing decision. **The Directory** is edited by highly renowned OOP industry experts, Jim Salmons and Timlynn Babitsky. Its organized format directs you quickly to the section you're looking for. Complete cross-referencing helps you locate valuable information faster and easier than ever before.

Here's what's in it for you!

- ▲ **Company Section** — A-Z listing of over 200 OOP-related companies. Synopses of companies' backgrounds, contacts and products disclose vital corporate information. Cross-referenced index by product name, platforms supported and base languages used.
- ▲ **Product Section** — alphabetized by Product Category for nearly 300 products — cross-referenced indexes to base language, operating system or platform, product category and product price. Includes insightful published product reviews.
- ▲ **Consultants Section** — alpha listing of contacts and specialties.

OOPS in Print Update

- ▲ *Selected* reprints of landmark articles from the ***Journal of Object-Oriented Programming, The C++ Report*** and the ***HOTLINE on Object-Oriented Technology.***
- ▲ **Articles** — citation listing of published articles cross-referenced by language and major subject area.
- ▲ **Books** — Complete listing of published books and in-print books — listed by author and subject.
- ▲ **Conferences/Seminars/Training** — alphabetized list by sponsoring organization with dates, courses, length, price and contact to help plan your calendar and budgets. Included are upcoming and regularly recurring events.

The 1990 International OOP Directory assists you in —

- ▲ Quickly finding exactly what you need.
- ▲ Sharpening your budget with all product pricing information at your fingertips.
- ▲ Locating the products compatible with your operating system.
- ▲ Spotting up-to-the-minute trends in the marketplace.
- ▲ Contacting those software manufacturers you want to do business with.
- ▲ Reviewing pertinent landmark articles you may have missed. Using the spectrum of published articles as your handy "clipping service."
- ▲ Getting product reviews that can save you time, money and inconvenience.
- ▲ Scanning years of costly research from important selected articles.
- ▲ Cutting through the clutter to determine the books you want to read.

Stop wasting time with limited "bits and pieces" of information from more generally focused software magazines. Simply turn to the only directory that tunes you into the entire object-oriented field. Now you can easily access OOP information whenever you need it . . . directly.

This "source book to the O-O industry" is like having a well-connected consultant at your disposal — Anytime. Full-time. The simple fact is you'll find everything that has anything to do with object-oriented technology right here. Your satisfaction is guaranteed.

Simply purchase the **1990 International OOP Directory** at no risk. If you are not *100% satisifed* for any reason, return it for a full refund within 15 days. No questions asked.

Circle **23** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

New Turbo C++ Professional

When the object is programming

C++

Be Objective

Object-Oriented Programming (OOP) *is* programming in the '90s. It's the next step after structured programming and is the best way to write applications. So Borland combined the power of OOP with the efficiency of C to produce new Turbo C++ Professional.

And Turbo C++ Professional is the first Turbo-charged native code C++ compiler that brings Object-Oriented Programming to your PC. Since Turbo C++ Professional also compiles ANSI C code, you can be productive with C now, and move to C++ at your own pace.

Environment ++

The best compiler deserves the best environment, and our new Programmer's Platform™ environment makes you more productive. It features overlapping windows and mouse support. And sports a new multi-file editor, an integrated debugger, and a smart project manager. Its advanced open architecture lets you integrate the tools you need to feel right at home.

VROOMM adds room

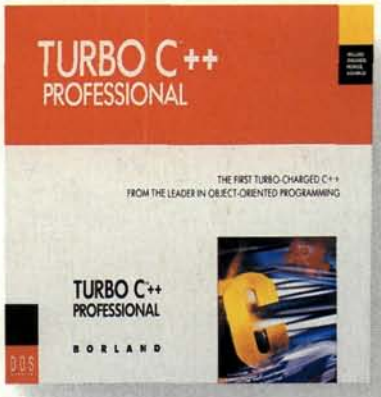
VROOMM™ (Virtual Runtime Object-Oriented Memory Manager) lets you break the 640K barrier. Just select the application code you want to overlay, and VROOMM does the rest—swapping modules on demand. It's fast, easy, automatic.

Another +

Turbo C++ Professional gives you all the tools you need to build fast, reliable C++ programs.

Turbo Debugger® 2.0 debugs your object-oriented programs. This powerful new version is the first and only debugger to support *reverse execution*. Letting you step backwards through your code to find the bugs you might have missed.

New Turbo Profiler,™ the world's first interactive profiler, displays histograms of your program's performance. With it, you can easily spot execution bottlenecks, and see where improvements or redesign of your code will yield maximum performance gains.



And Turbo Assembler® 2.0 lets you replace time-critical segments of your code using the world's fastest MASM-compatible assembler.

Turbo C++ Professional Compiler
- C++ conforming to AT&T's 2.0 specification
- C++ class libraries
- Full ANSI C compiler
- VROOMM overlay manager
- Complete documentation and tutorials

Programmer's Platform
- Open architecture for integration of your own tools
- Overlapping windows with mouse support
- Multifile, macro-based editor
- Smart project manager provides visual MAKE
- Integrated debugging and hypertext help

Turbo Debugger 2.0
- Class hierarchy browser and inspectors
- Reverse execution provides "true" undo
- 286 protected-mode and 386 virtual-mode debugging
- Keystroke record and playback

NEW Turbo Profiler
- Displays histograms of program execution
- Tracks call history, overlays, interrupts, file I/O

Turbo Assembler 2.0
- Multipass assembler with NOP squishing and 486 support

Special Introductory Offer

The suggested retail price for Turbo C++ Professional is $299.95 ($199.95 for Turbo C++). For a limited time, Borland is offering its dealers and distributors special introductory discounts.* So be objective, and **SEE YOUR DEALER** or call Borland** at **1-800-331-0877** now!

BORLAND

Code: MC67

*Offer expires July 31, 1990 or while supplies last. Offer good in United States and Canada only. **Special discounts for registered Turbo C® owners are available from Borland. Mail orders to: Borland, P.O. Box 660001, Scotts Valley, CA 95067-0001. For orders outside the U.S., call (408) 438-5300. Turbo C++, Turbo Debugger, Turbo Profiler and Turbo Assembler are trademarks or registered trademarks of Borland International, Inc. Copyright © 1990, Borland International, Inc. All rights reserved. BI-1333

Circle 3 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED